Capítulo

4

Arquiteturas para Redes de Sensores Sem Fio[1]

Linnyer Beatrys Ruiz[1,2], Luiz Henrique A. Correia[1,3], Luiz Filipe M. Vieira[1], Daniel F. Macedo[1], Eduardo F. Nakamura[1,4], Carlos M. S. Figueiredo[1,4] Marcos Augusto M. Vieira[1], Eduardo Habib Bechelane [1], Daniel Camara[1], Antonio A.F. Loureiro[1], José Marcos S. Nogueira[1], Diógenes C. da Silva Jr.[5]

[1]Departamento de Ciência da Computação da UFMG

[2]Departamento de Informática da PUCPR

[3]Departamento de Ciência da Computação da UFLA

[4]Fundação Centro de Análise, Pesquisa e Inovação Tecnológica - FUCAPI

[5]Departamento de Engenharia Elétrica da UFMG

Abstract

Wireless Sensor Networks (WSNs) are a special kind of a MANET (Mobile Ad hoc Network) and play an important role in the ubiquitous computing. WSNs are expected to have a large number of autonomous devices called sensor nodes. The main objective of a WSN is to monitor and, eventually, control an environment, in general, without human intervention. Sensor nodes collect data about some physical phenomenon, process data in the node, and disseminate data using, for instance, multi-hop communication. A WSN tends to be application-dependent, i.e., the hardware and software requirements and the operation modes vary according to the application. This chapter introduces the main concepts related to WSNs, presents a classification scheme for this kind of network, and describes the basic functionalities of the main communication protocols published in the literature for different kinds of applications. Furthermore, this chapter presents the main architectures (platforms) for sensor nodes developed by different research groups, with emphasis for the Mica Motes platform that incorporates the TinyOs operating system. Finally, the chapter discusses the development process of an application based on the Mica Mote, TinyOs, TOSSIM simulator, and the TinyViz interface.

[1]Este trabalho foi financiado parcialmente pelo CNPq.

Resumo

Redes de Sensores Sem Fio (RSSFs) são um tipo especial de rede móvel ad-hoc e têm um papel importante na computação ubíqüa. Em geral, as RSSFs são formadas por um grande número de dispositivos autônomos chamados nós sensores. RSSFs têm como objetivo monitorar e, eventualmente, controlar um ambiente, normalmente, sem intervenção humana direta. Os nós sensores coletam dados sobre fenômenos de interesse, realizam processamento local e disseminam os dados usando, por exemplo, comunicação multi-saltos. Uma RSSF tende a ser dependente da aplicação a que se destina, isto é, os requisitos de hardware e software e os mecanismos de operação variam de acordo com a aplicação. Este capítulo tem por objetivo introduzir os principais conceitos de RSSFs, apresentar um esquema de classificação para essas redes e descrever as funcionalidades básicas dos principais protocolos de comunicação publicados na literatura para diferentes tipos de aplicação. Além disso, o capítulo apresenta as principais arquiteturas (plataformas) de nós sensores sem fio desenvolvidas por diferentes grupos de pesquisa dando ênfase à plataforma Mica Motes, que utiliza o ambiente operacional TinyOs. O capítulo também discute o processo de desenvolvimento de uma aplicação usando o nó Mica Mote, o ambiente TinyOs, o simulador TOSSIM e a interface de visualização TinyViz.

4.1. Introdução

Redes de Sensores Sem Fio (RSSFs) têm sido viabilizadas pela rápida convergência de três tecnologias: microprocessadores, comunicação sem fio e micro sistemas eletro-mecânicos (MEMS – *Micro Electro-Mechanical Systems*) [Loureiro et al., 2002]. Uma RSSF pode ser usada para monitorar e, eventualmente, controlar um ambiente. Este tipo de rede é formado geralmente por centenas ou milhares de dispositivos autônomos que tendem a ser projetados com pequenas dimensões (cm^3 ou mm^3) chamados nós sensores. Os principais componentes de um nó sensor são transceptor para comunicação sem fio, fonte de energia, unidade de sensoriamento, memória e processador. O componente lógico de um nó sensor é o software que executa no processador [Loureiro et al., 2003]. Existem casos em que uma RSSF também pode ser composta de dispositivos chamados atuadores que permitem ao sistema controlar parâmetros do ambiente monitorado.

Os nós individualmente possuem pouca capacidade computacional e de energia, mas um esforço colaborativo entre os mesmos permite a realização de uma grande tarefa [Ruiz et al., 2004]. Os nós sensores podem ser lançados sobre áreas remotas (reservas ambientais, oceanos, vulcões, rios, florestas, etc.) e, sem intervenção de técnicos ou operadores, formar uma rede sem fio *ad hoc* que coleta dados sobre os fenômenos de interesse, realiza processamento local, e dissemina as informações para um ponto de acesso em um esquema de comunicação multi-saltos (*multi-hop*). O ponto de acesso é o elemento através do qual a rede comunica-se com outras redes ou com um ou mais observadores [Ruiz et al., 2003]. O ponto de acesso pode ser implementado em um nó sensor que será chamado de nó sorvedouro (*sink node*) ou em uma estação base (BS - *Base Station*).

RSSFs podem ser vistas como um tipo especial de rede móvel *ad hoc* (MANET - *Mobile Ad hoc Network*) [Loureiro et al., 2003] e como uma das vertentes da computação

ubíqüa. Em breve elas estarão conectadas à Internet [SensorNet, 2003]. RSSFs diferem de redes de computadores tradicionais em vários aspectos. Em geral, as RSSFs possuem um grande número de elementos distribuídos, operam sem intervenção humana direta, têm restrições severas de energia, e devem possuir mecanismos para auto-gerenciamento (auto-configuração, auto-manutenção, auto-organização, auto-proteção etc.) devido a deposição em áreas remotas, a pouca capacidade individual dos nós e a topologia dinâmica. Os nós de uma RSSF podem ser descartados, perdidos ou saírem de serviço por diferentes razões como falta de energia, problemas na deposição, ameaças e ataques a segurança, falhas nos componentes e falha de comunicação [Ruiz, 2003]. Mesmo sem a mobilidade dos nós, a topologia da rede é dinâmica. Algoritmos distribuídos tradicionais, como protocolos de comunicação e eleição de líder devem ser revistos para esse tipo de ambiente antes de serem usados diretamente [Loureiro et al., 2003].

Do ponto de vista científico, as RSSFs apresentam uma grande variedade de novos problemas ainda não estudados ou ainda incipientes. No relatório do Workshop sobre pesquisas fundamentais na área de redes, patrocinado pela *National Science Foundation* (NSF) [National Science Foundation, 2004], a pesquisa em redes de sensores foi considerada uma das seis áreas de grande desafio de pesquisa. Outra área relacionada foi a de teoria de redes de comunicação sem fio, que engloba RSSFs.

O projeto de uma RSSF é influenciado por muitos fatores que incluem tolerância a falhas, escalabilidade, custo de produção, ambiente operacional, topologia da rede, restrições de hardware, meio de transmissão e consumo de energia. Cada um destes fatores exige requisitos específicos na concepção e projeto dos nós, assim como em todas as camadas da pilha de protocolos. Esquemas de modulação, estratégias para superar os efeitos da propagação de sinal e projeto de hardware de baixo consumo são requisitos do projeto da camada física. Determinação do limite inferior de energia requerida para auto-organização da rede, esquemas de controle de erro, modos de operação para economizar energia e cuidados com a mobilidade são os desafios da camada de enlace de protocolos de controle de acesso ao meio. Tratar das mudanças de topologia, endereçamento, escalabilidade e interface com outras redes são requisitos esperados para a camada de rede. Já os protocolos da camada de transporte devem permitir a diversidade de comunicação fim-a-fim para calcular variações nas características do canal de comunicação [National Science Foundation, 2004]. O processamento local dos dados através de correlação (fusão, contagem, agregação, compressão, etc.) também são requisitos a serem considerados no projeto dos protocolos para disseminação e consulta aos nós sensores, assim como os requisitos de segurança em cada uma das camadas da pilha de protocolos.

Todos os fatores citados acima são influenciados pelos requisitos da aplicação, isto porque, uma RSSF é um tipo de sistema dependente da aplicação. Os parâmetros de configuração, operação e manutenção variam com os objetivos da aplicação. Protocolos para cada uma das camadas da pilha têm sido propostos na literatura. No projeto desses protocolos, alguns dos fatores citados são considerados, assim como a dependência da aplicação. No escopo deste texto, estão descritas as funcionalidades básicas dos principais protocolos disponíveis na literatura para as camadas de enlace, rede e transporte. Além dos protocolos

para cada uma dessas camadas, este documento também apresenta os principais conceitos e ferramentas necessárias ao desenvolvimento de aplicações para RSSFs, considerando o ambiente de um nó sensor comercialmente disponível, o Mica Motes [Motes, 2002].

O texto está organizado como descrito a seguir. A seção 4.2 apresenta um esquema de caracterização das RSSFs de acordo com os requisitos da aplicação. A seção 4.3 apresenta os principais protocolos propostos na literatura para as camadas de enlace, rede e transporte da pilha de protocolos. A seção 4.4 apresenta algumas plataformas e projetos acadêmicos de nós sensores sem fio. A seção 4.5 apresenta o sistema operacional TinyOs desenvolvido para processadores utilizados em nós Mica Motes. A seção 4.6 trata do desenvolvimento de aplicações para RSSFs a partir da arquitetura de nós Mica Motes e do sistema operacional TinyOS. A seção 4.7 descreve os principais passos a serem seguidos no desenvolvimento de uma aplicação para um nó Mica Motes usando o sistema operacional TinyOS. As considerações finais são apresentadas na seção 4.8.

4.2. Caracterização das Redes de Sensores Sem Fio

A classificação de uma RSSF depende de seu objetivo e área de aplicação. A aplicação influenciará diretamente nas funções exercidas pelos nós da rede, assim como na arquitetura desses nós (processador, memória, dispositivos sensores, fonte de energia, transceptor), na quantidade de nós que compõem a rede, na distribuição inicialmente planejada para a rede, no tipo de deposição dos nós no ambiente, na escolha dos protocolos da pilha de comunicação, no tipo de dado que será tratado, no tipo de serviço que será provido pela rede e conseqüentemente no tempo de vida dessa rede.

De acordo com [Ruiz, 2003], as RSSFs podem ser classificadas segundo a configuração (ver tabela 4.1), o sensoriamento (ver tabela 4.2) e segundo o tipo de comunicação (ver tabelas 4.3 e 4.4). Uma RSSF também pode ser diferente segundo o tipo de processamento que executa (ver tabela 4.5).

O potencial de observação e controle do mundo real permite que as RSSFs se apresentem como uma solução para diversas aplicações: monitoração ambiental, gerenciamento de infra-estrutura, biotecnologia, monitoração e controle industrial, segurança pública e de ambientes em geral, áreas de desastres e risco para vidas humanas, transporte, medicina e controle militar [Badrinath et al., 2000, Estrin et al., 2000], [Lindsey et al., 2002, Meguerdichian et al., 2001, Srivastava et al., 2001]. A visão é que RSSFs se tornem disponíveis em todos os lugares executando as tarefas mais diferentes possíveis [National Science Foundation, 2004]. Este potencial tem estimulado ainda mais o desenvolvimento de hardware e software para RSSFs e atraído a atenção da comunidade acadêmica.

Como mencionado antes, uma RSSF é um tipo de sistema dependente da aplicação. Qualquer projeto ou solução proposta para estas redes deve levar em consideração os requisitos da aplicação a ser desenvolvida, as características e restrições dos componentes dos nós sensores, assim como as características do ambiente onde tais redes serão aplicadas.

Configuração		
Composição	Homogênea	Rede composta de nós que apresentam a mesma capacidade de hardware. Eventualmente os nós podem executar software diferente.
	Heterogênea	Rede composta por nós com diferentes capacidades de hardware.
Organização	Hierárquica	RSSF em que os nós estão organizados em grupos (*clusters*). Cada grupo terá um líder (*cluster-head*)que poderá ser eleito pelos nós comuns. Os grupos podem organizar hierarquias entre si.
	Plana	Rede em que os nós não estão organizados em grupos
Mobilidade	Estacionária	Todos os nós sensores permanecem no local onde foram depositados durante todo o tempo de vida da rede.
	Móvel	Rede em que os nós sensores podem ser deslocados do local onde inicialmente foram depositados.
Densidade	Balanceada	Rede que apresenta uma concentração e distribuição de nós por unidade de área considerada ideal segundo a função objetivo da rede.
	Densa	Rede que apresenta uma uma alta concentração de nós por unidade de área.
	Esparsa	Rede que apresenta uma baixa concentração de nós por unidade de área.
Distribuição	Irregular	Rede que apresenta uma distribuição não uniforme dos nós na área monitorada.
	Regular	Rede que apresenta uma distribuição uniforme de nós sobre a área monitorada

Tabela 4.1: Caracterização das Redes de Sensores Sem Fio segundo a configuração.

Sensoriamento		
Coleta	Periódica	Os nós sensores coletam dados sobre o(s) fenômeno(s) em intervalos regulares. Um exemplo são as aplicações que monitoram o canto dos pássaros. Os sensores farão a coleta durante o dia e permaneceram desligados durante a noite.
	Contínua	Os nós sensores coletam os dados continuamente. Um exemplo são as aplicações de exploração interplanetária que coletam dados continuamente para a formação de base de dados para pesquisas.
	Reativa	Os nós sensores coletam dados quando ocorrem eventos de interesse ou quando solicitado pelo observador. Um exemplo são as aplicações que detectam a presença de objetos na área monitorada.
	Tempo Real	Os nós sensores coletam a maior quantidade de dados possível no menor intervalo de tempo. Um exemplo são aplicações que envolvem risco para vidas humanas tais como aplicações em escombros ou áreas de desastres. Um outro exemplo são as aplicações militares onde o dado coletado é importante na tomada de decisão e definição de estratégias.

Tabela 4.2: Caracterização das Redes de Sensores Sem Fio segundo o sensoriamento.

4.3. Arquitetura de Comunicação

Como mencionado, as RSSFs têm características particulares e o uso direto de protocolos de comunicação de redes ad hoc não é viável, pois estes requerem muitos Kbytes de memória e

Classificação segundo a Comunicação		
Disseminação	Programada	Os nós disseminam em intervalos regulares.
	Contínua	Os nós disseminam os dados continuamente.
	Sob Demanda	Os nós disseminam os dados em resposta à consulta do observador e à ocorrência de eventos.
Tipo Conexão	Simétrica	Todas as conexões existentes entre os nós sensores, com exceção do nó sorvedouro têm o mesmo alcance.
	Assimétrica	As conexões entre os nós comuns têm alcance diferente.
Transmissão	Simplex	Os nós sensores possuem transceptor que permite apenas transmissão da informação.
	Half-duplex	Os nós sensores possuem transceptor que permite transmitir ou receber em um determinado instante.
	Full-duplex	Os nós sensores possuem transceptor que permite transmitir ou receber dados ao mesmo tempo.

Tabela 4.3: Caracterização das Redes de Sensores Sem Fio segundo a comunicação (Parte A).

Classificação segundo a Comunicação		
Alocação de Canal	Estática	Neste tipo de rede se existirem "n" nós, a largura de banda é dividida em "n" partes iguais na freqüência (FDMA – *Frequency Division Multiple Access*), no tempo (TDMA – *Time Division Multiple Access*), no código (CDMA – Code Division Multiple Access), no espaço (SDMA – *Space Division Multiple Access*) ou ortogonal (OFDM – *Orthogonal Frequency Division Multiplexing*). A cada nó é atribuída uma parte privada da comunicação, minimizando interferência.
	Dinâmica	Neste tipo de rede não existe atribuição fixa de largura de banda. Os nós disputam o canal para comunicação dos dados.
Fluxo de Informação	*Flooding*	Neste tipo de rede, os nós sensores fazem *broadcast* de suas informações para seus vizinhos que fazem *broadcast* desses dados para outros até alcançar o ponto de acesso. Esta abordagem promove um alto *overhead* mas está imune às mudanças dinâmicas de topologia e a alguns ataques de impedimento de serviço (DoS – *Denial of Service*).
	Multicast	Neste tipo de rede os nós formam grupos e usam o *multicast* para comunicação entre os membros do grupo.
	Unicast	Neste tipo de rede, os nós sensores podem se comunicar diretamente com o ponto de acesso usando protocolos de roteamento multi-saltos.
	Gossiping	Neste tipo de rede, os nós sensores selecionam os nós para os quais enviam os dados.
	Bargaining	Neste tipo de rede, os nós enviam os dados somente se o nó destino manifestar interesse, isto é, existe um processo de negociação.

Tabela 4.4: Caracterização das Redes de Sensores Sem Fio segundo a comunicação (Parte B).

Classificação segundo o Processamento		
Cooperação	Infra-estrutura	Os nós sensores executam procedimentos relacionados à infra-estrutura da rede como por exemplo, algoritmos de controle de acesso ao meio, roteamento, eleição de líderes, descoberta de localização e criptografia.
	Localizada	Os nós sensores executam além dos procedimentos de infra-estrutura, algum tipo de processamento local básico como por exemplo, tradução dos dados coletado pelos sensores baseado na calibração.
	Correlação	Os nós estão envolvidos em procedimentos de correlação de dados como fusão, supressão seletiva, contagem, compressão, multi-resolução e agregação.

Tabela 4.5: Caracterização das Redes de Sensores Sem Fio segundo o processamento.

vários recursos. Novos protocolos têm sido desenvolvidos para se adequar às necessidades e limitações das RSSFs. Nesta seção são apresentados os protocolos de comunicação desenvolvidos especificamente para RSSFs. O estudo desses protocolos pode ser feito por camadas como sugerido pela arquitetura TCP/IP e mostrado na tabela 4.6.

4.3.1. Camada Física

Em uma RSSF podem ser exploradas três possibilidades para comunicação sem fio: ótica, infra-vermelho e Rádio Freqüência (RF).

A comunicação ótica consome menor quantidade de energia por bit transmitido e não requer área física para instalação de antena, mas necessita de uma linha de sinal (LOS - *Line of Sight*) para comunicação, isto é, transmissor e receptor devem estar alinhados. A comunicação direcional não é viável nas aplicações em que os nós são lançados sobre a área monitorada. Além disso, a comunicação ótica é sensível às condições atmosféricas. Um exemplo da utilização de comunicação ótica é o nó sensor Smart Dust [Dust, 2002], onde a comunicação ótica pode ser passiva, através de um *Corner Cube Reflector* (CCR) (0,5 $\times$ 0,5 $\times$ 0,1 mm^3) transmitindo a uma taxa de 10 kbps, utilizando 1 μW de energia e com uma área de alcance de 1km. Outra opção no Smart Dust é a transmissão ativa através de laser, (1,0 $\times$ 0,5 $\times$ 0,1 mm^3) transmitindo a 1 Mbps, com o gasto de energia de 10 mW e área de alcance de 10km. O volume total de um nó sensor Smart Dust chega a 1,5 mm^3 e a massa total 5mg, dimensões que tornam inviável o uso de transceptor de RF.

A comunicação através de infra-vermelho também é usualmente direcional e tem alcance de um metro. A vantagem no caso da comunicação infra-vermelho é não precisar de área física para antena, contudo ainda não estão disponíveis nós que utilizem este tipo de comunicação.

A comunicação em RF é baseada em ondas eletromagnéticas e um dos maiores desafios para o uso deste tipo de comunicação em RSSFs é o tamanho da antena. Para otimizar a transmissão e a recepção, uma antena deve ser pelo menos "$B/4$", onde B é o comprimento de onda da freqüência. Assumindo um nó sensor em que um quarto do comprimento de onda será 1 mm, a freqüência do rádio seria de 75 GHz. Também é necessário reduzir o consumo

de energia com modulação, filtragem, demodulação, etc. As vantagens da comunicação em RF são a facilidade de uso e a aceitação comercial, que tornam este tipo de comunicação viável para plataformas de nós sensores. Vários aspectos afetam o consumo de energia do rádio, incluindo tipo de modulação, taxa de dados e energia de transmissão. Em geral, os rádios podem operar em quatro modos distintos: transmitindo, recebendo, "*idle*" e "*sleep*". Muitos rádios que operam no modo "*idle*" consomem energia como se estivessem no modo de recepção, nestes casos é importante traçar outras estratégias para economia de energia [Vieira et al., 2003].

Dois modelos de rádio têm sido usados comercialmente em nós sensores: TR1000 [TR 1000, 2004] e o CC1000 [CC 1000, 2004]. O modelo TR é um transceptor de rádio híbrido que suporta transmissão de dados em taxas superiores a 115.2 kbps, com alcance de 30 a 90 metros e opera em 3V. Com estas taxas ele consome aproximadamente 14.4 mW na recepção, 36 mW durante a transmissão e 15 μW no modo "*sleep*". O rádio Chipcon CC1000 é um transceptor CMOS RF de baixo consumo de energia que obtém transferência de dados de até 76.8 kbps. O CC1000 foi projetado para modulação FSK (*Frequency Shift Key*) na faixa de banda ISM (*Industry Science Medical*) de 315, 433, 868 e 915 MHz. No modo de baixo consumo, a corrente consumida é de 0.2 μA. A tensão de operação varia de 2.1 a 3.6 V.

Outro exemplo de transceptor é o módulo de rádio do nó sensor WINS (ver seção 4.4.5), o Conexant RDSSS9M que implementa uma comunicação RF *spread spectrum* a uma freqüência de 900 MHz (ISM *Industrial Scientific Medical*). O rádio opera em um dos 40 canais, escolhido pelo controlador. O rádio é capaz de operar a vários níveis de energia para transmissão, podendo variar de 1 mW até 100 mW, permitindo assim o uso de algoritmos de otimização do consumo de energia para a transmissão.

Camada	**Protocolos**
Transporte	PFSQ, ESRT, RMST
Rede	DD, SPIN, SAR, MULTI, STORM, PROC, TinyBeaconing, LEACH, LEACH-C, TEEN, PEGASIS, ICA, GEOMOTE, GEAR, GPSR
Enlace	S-MAC, ARC, T-MAC, B-MAC,DE-MAC, TRAMA
Física	Transmissão em Rádio Freqüência (RF), ótica e infra-vermelho

Tabela 4.6: Protocolos para RSSFs

4.3.2. Camada de Enlace

Os requisitos da camada de enlace são diferentes para os diferentes tipos de RSSF apresentados nas tabelas 4.1, 4.2, 4.3 e 4.4. Por exemplo, os nós de uma RSSF sob demanda podem permanecer com os transceptores inativos por longos períodos de tempo e que repentinamente tornam-se ativos quando algum fenômeno é detectado. Se a RSSF for densa, vários nós sensores na área de ocorrência do evento estarão acessando o meio ao mesmo tempo para transmitir os dados. As características particulares das RSSFs e sua dependência da aplicação motivam um controle de acesso ao meio (MAC - *Medium Access Control*) que é diferente do tradicional tal como o IEEE 802.11 [IEEE 802.11, 2003]. Quase sempre a conservação de energia e a auto-organização são objetivos primários.

As soluções existentes para métodos de acesso ao canal em redes *ad hoc* podem ser divididas em duas categorias, baseadas em contenção e métodos organizados (ver item alocação de canal na tabela 4.4). Os métodos baseados em contenção são um problema para redes que continuamente sentem o canal de acesso, por exemplo uma RSSF de disseminação contínua e tempo real (ver tabela 4.3), e com isso perdem recursos sempre que uma colisão ocorre. Nas RSSFs hierárquicas (ver tabela 4.1), os métodos organizados de acesso ao canal tentam primeiro determinar a conectividade entre os nós e então manipulam a atribuição de canais (*slots*) de maneira hierárquica formado grupos de nós e designando líderes para o grupo.

As RSSFs são diferentes das redes tradicionais mas herdaram os problemas de comunicação das redes sem fio. Na maior parte dos casos, as redes sem fio empregam um rádio de um único canal com modo de comunicação *half-duplex*, ou seja, a comunicação é bidirecional e não simultânea. O rádio utilizando a mesma freqüência pode somente transmitir ou receber informações a cada instante de tempo (ver tabela 4.4). Sendo assim, o método empregado nas tradicionais redes Ethernet, CSMA/CD (*Carrier Sense Multiple Access with Collision Detect*) não pode ser empregado em redes sem fio [Tanenbaum, 2003].

As redes sem fio utilizam o protocolo CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance)* para controle de acesso ao meio que evita a ocorrência de colisões. O CSMA/CA utiliza um diálogo de três passos: *RTS-CTS-DATA*[2] envolvendo a comunicação entre as duas estações transmissora e receptora. É importante observar que nas redes sem fio as colisões ocorrem somente no receptor, já que a estação base ao transmitir não tem como escutar o canal. As colisões podem ocorrer na recepção de pacotes de controle e de dados. As estações na rede ao escutarem pacotes de controle RTS ou CTS não destinados a elas devem bloquear seus rádios até o final da transmissão. Este procedimento diminui a probabilidade de ocorrência de colisões na rede. As colisões em redes sem fio também ocorrem por um problema conhecido como terminal escondido: Uma estação A transmite seu RTS para uma estação B dentro de seu alcance de rádio. Uma outra estação C, que está dentro do alcance de rádio de B, mas fora do alcance de A, também envia um RTS para a estação B. Para esta situação haverá colisão na estação B como mostrado na figura 4.1.

Outra dificuldade comum em redes sem fio é o problema da estação exposta: A estação B solicita transmissão à estação A enviando um pacote de controle RTS. Neste momento, a estação C está pronta para transmitir, mas como ela está dentro do alcance do rádio de B, ela escuta o pacote de controle e bloqueia seu rádio até que a transmissão termine. Se a estação C deseja transmitir para uma estação diferente de B, por exemplo, para a estação D fora do alcance de B, ela estará impedida de transmitir. A transmissão da estação C para a estação D não irá interferir na comunicação entre A e B, então a escuta do pacote RTS não fornece informação completa. Neste caso, dizemos que a estação C está exposta às transmissões da estação B, conforme mostrado na figura 4.2.

As restrições dos protocolos empregados em RSSFs são ainda maiores que as restrições das redes MANETs, devido ao hardware empregado. Desta forma, não existe suporte pelo hard-

[2] *Request-To-Send—Clear-To-Send-Data*

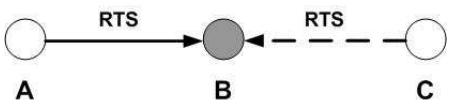


Figura 4.1: Problema do terminal escondido: C está escondido de A.

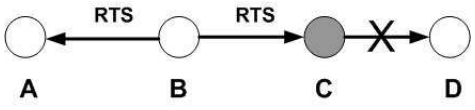


Figura 4.2: Problema da estação exposta: C está exposta para B.

ware para detecção de portadora, detecção de colisão, enquadramento específico, codificação ou balanceamento de energia. O rádio utilizado possui características de baixa potência, largura de banda limitada e um único canal na freqüência base ISM.

Os principais protocolos de acesso ao meio projetados para RSSFs e disponíveis na literatura atual são descritos a seguir.

4.3.2.1. Protocolo S-MAC

O S-MAC (*Sensor-MAC*) [Ye et al., 2002] é um protocolo de controle de acesso ao meio baseado em alocação dinâmica de canal, mas que utiliza sincronização para coordenação dos modos de operação do rádio. É destinado a redes com aplicações dirigidas a eventos, com coleta periódica de dados, insensíveis à latência e com baixa taxa de envio de mensagens. A comunicação entre os nós segue um fluxo *broadcast* ou um fluxo *unicast* para troca de mensagens. Considera os requisitos de uma rede densa e homogênea para ser eficiente em energia e permitir a autoconfiguração dos nós da rede. O protocolo S-MAC procura ser eficiente em energia reduzindo o consumo dos principais eventos responsáveis pelo desperdício de energia descritos a seguir:

- Colisões: os nós desejam transmitir ao mesmo tempo para um mesmo destino. Para resolver o problema de colisão o S-MAC emprega a mesma técnica utilizada no IEEE 802.11—DCF (*Distributed Coordination Function*) [Standard Commitee of IEEE Computer Society, 1999], usando um diálogo de comunicação *RTS-CTS-DATA-ACK*. Este diálogo de comunicação evita colisões, problemas de terminal escondido e problema de estação exposta. Caso ocorra a colisão utiliza um algoritmo para aguardar um tempo aleatório, o BEB (*Binary Exponential Backoff*).
- *Overhearing*: os nós escutam transmissões de pacotes destinados a outros nós. A técnica empregada pelo S-MAC é desligar o rádio do nó (modo *sleep*) quando verifica que o pacote não é destinado a ele.
- *Overhead*: os pacotes de controle são utilizados para reserva do canal de comunicação, reconhecimento de pacotes de dados, sincronização e outros. Estes pacotes de controle aumentam o tráfego da rede, mas não transportam dados úteis. O S-MAC reduz o tamanho dos pacotes de controle para diminuir o *overhead.*
- *Idle listening*: o nó fica escutando o meio mesmo quando não existe tráfego na rede. O S-MAC utiliza um ciclo de operação reduzido com tempos fixos de atividade (*listen*) e

de repouso (*sleep*). O tempo de atividade é menor que o tempo de repouso (cerca de 10%).

A sinalização para os pacotes de controle e de sincronização é feita dentro do canal, enviando um pacote SYNC em *broadcast* para todos os seus vizinhos. O S-MAC aplica a técnica de *message passing* para reduzir a latência durante a contenção em aplicações que requerem armazenamento de informações para processamento na rede (*in-network*). Esta técnica permite a transmissão de mensagens longas, que são divididas em pequenos fragmentos e enviadas em rajada. Este protocolo obtém considerável redução do consumo de energia, prolonga o tempo de vida da rede e encontra-se implementado na plataforma Mica Motes.

4.3.2.2. Protocolo ARC

O ARC (*Adaptive Rate Control*) [Woo and Culler, 2001] tem como metas a alocação de largura de banda, justiça e eficiência em energia para condições de tráfego alto e baixo da rede. O protocolo propõe um mecanismo que passivamente adapta a taxa de transmissão dos dois tipos de tráfego: de passagem e de dados originados. Este mecanismo usa um incremento linear e um decremento multiplicativo para controlar a taxa de dados. A capacidade computacional exigida dos nós por este esquema é pequena e está dentro das limitações de hardware. Um novo esquema CSMA é proposto pelo ARC, adicionando-se um atraso aleatório antes do tempo de escuta, para evitar repetidas colisões devido ao comportamento sincronizado do nó na ocorrência de um evento. Este esquema CSMA é composto pelas seguintes fases: atraso inicial aleatório, tempo de escuta – intervalo fixo de tempo, e mecanismo de *backoff* – tempo de atraso gerado com janela fixa, com incremento binário exponencial ou com decremento binário exponencial.

O ARC em conjunto com este novo mecanismo CSMA fornece controle efetivo de acesso ao meio sem a utilização de pacotes explícitos de controle. Este esquema encontra justiça e mantém razoável largura de banda, sendo eficiente em energia para situações de baixo tráfego.

4.3.2.3. Protocolo T-MAC

O protocolo Time-out-MAC é baseado em contenção para o controle de acesso ao meio em RSSFs [van Dam and Langendoen, 2003]. O T-MAC foi desenvolvido para aplicações dirigidas a eventos que possuem baixa taxa de entrega de mensagens, insensíveis a latência e com transmissão contínua ou periódica de dados. A meta do T-MAC é ser eficiente em energia, considerando as limitações do hardware do nó e os padrões de comunicação de troca de mensagens entre seus vizinhos e entre os nós e a estação base.

O ciclo de operação é reduzido e possui tempos de atividade (*listen*) e de repouso *(sleep)* variáveis que se adaptam à carga da rede. A variação dinâmica do tempo ativo é obtida pela implementação de um temporizador que desliga o rádio do nó ao verificar que não existe transmissão durante um intervalo de tempo, conforme mostrado na figura 4.3, que descreve o ciclo adaptativo do protocolo T-MAC, onde as setas indicam transmissão e recepção de mensagens.

A idéia do T-MAC é reduzir o tempo de *idle listening* para diminuir o consumo de energia do nó. As mensagens recebidas durante o tempo de repouso são armazenadas e transferidas em rajadas no início do tempo ativo.

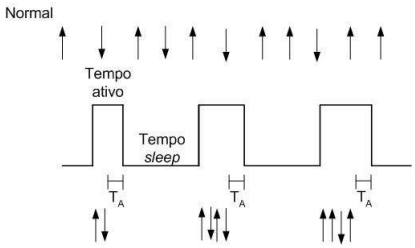


Figura 4.3: T-MAC.

O nó escuta a rede, transmite e recebe dados durante seu tempo ativo. O temporizador determina o final do tempo ativo quando não ocorrem eventos durante um tempo T_A. A ativação por eventos ocorre por: início periódico de quadro, recepção de dados no rádio, final da transmissão de seus vizinhos, final da transmissão de seu próprio pacote de dados ou recebimento de ACK, ou por detecção de sinal no rádio (*RSSI - Received Signal Strenght Indicator*). Os nós se comunicam com o diálogo *RTS-CTS-DATA-ACK* para evitar colisões e obter transmissão confiável. De maneira semelhante ao S-MAC, o T-MAC utiliza agrupamentos virtuais que seguem escalas para sincronizar seu ciclo de operação. Os nós transmitem suas escalas para os seus nós vizinhos através de pacotes SYNC.

A recepção de pacotes RTS ou CTS é suficiente para renovar o tempo T_A. O intervalo de tempo T_A deve ser suficiente para receber pelo menos o início de um pacote CTS, sendo obtido por:

$$T_A > t_{contencao} + t_{RTS} + RTT_{RTS}{}^{3}$$

O mecanismo de *backoff* é baseado em um número aleatório de intervalos fixos, calculados em função da carga máxima. Indiferentemente de sucesso ou falha na comunicação, a janela de contenção não é incrementada. Um problema é encontrado no T-MAC quando um nó dorme enquanto um outro nó ainda tem mensagem para ele. Este é conhecido como o problema de dormir cedo e pode ser visualizado pela figura 4.4, onde o nó D dorme antes de C enviar um RTS.

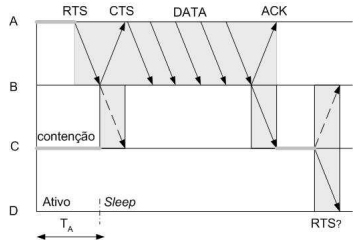


Figura 4.4: Dormir cedo.

Este problema pode ser resolvido de duas maneiras: (1) um nó ao escutar um pacote CTS destinado a outro nó envia imediatamente aos seus vizinhos um pacote designado de FRTS (*Future RTS*); (2) usar um esquema de priorizar o esvaziamento do *buffer* quando este estiver perto de sua capacidade limite. Um nó ao receber um RTS ao invés de responder com um CTS, transmite as mensagens armazenadas em seu *buffer* para o nó de destino. O T-MAC consegue ser mais eficiente em energia que o S-MAC, mas é extremamente limitado em largura de banda e o seu algoritmo não é aplicável depois que uma fração da largura de banda do canal é utilizada.

[3] $t_{contencao}$ - tamanho do intervalo de tempo de contenção; t_{RTS} - tamanho do pacote de RTS; RTT_{RTS} - é o tempo de transmissão de um pacote RTS (ida e volta).

4.3.2.4. Protocolo B-MAC

Este protocolo foi especificamente projetado para RSSFs e utiliza como plataforma de desenvolvimento o Mica Motes2 [Motes, 2002]. Encontra-se implementado na versão do TinyOS 1.1.3 como um novo método de CSMA/CA para RSSFs [Polastre, 2003].

A idéia do B-MAC é que, ao invés de inserir o algoritmo de *backoff* inicial dentro da camada MAC, seja estabelecida um política de gerenciamento em que a aplicação controle o *backoff* inicial antes de submeter um pacote para transmissão. O algoritmo de *backoff* binário exponencial não é usado para o controle de congestionamento, ao invés disso é verificado o estado do canal. O B-MAC utiliza a heurística chamada CCA (*Clear Channel Assessment*) para verificar se existe atividade no canal e para retornar a informação para a aplicação. O CCA emprega a técnica de julgamento de canal baseado em uma estimativa de ruído do canal obtida pela força do sinal recebido RSSI (*Received Signal Strenght Indicator*). Na implementação do B-MAC (TinyOS versão 1.1.3), o tamanho do preâmbulo das mensagens foi reduzido e o limite teórico do canal foi aumentado de 42 pacotes/segundo para 53 pacotes/segundo, considerando um tamanho de pacote de 36 bytes. O B-MAC é um novo método CSMA para RSSFs que encontra razoável vazão em comparação aos métodos tradicionais e proporciona boa taxa de utilização do canal, sendo flexível para diferentes aplicações.

4.3.2.5. Protocolo DE-MAC

O protocolo DE-MAC (*Distributed Energy aware MAC*) trata do gerenciamento e balanceamento de energia em RSSFs [Kalidindi et al., 2003]. É um protocolo que emprega alocação estática de canal TDMA, e portanto livre de colisões e de *overhead* de pacotes de controle. Utiliza o conceito de ciclo de operação reduzido com tempos de atividade (*listen*) e de repouso (*sleep*) para evitar o desperdício de energia com a escuta de pacotes destinados a outros nós (*overhearing*) e com a escuta do meio sem tráfego (*idle listening*).

O DE-MAC utiliza um algoritmo distribuído para balanceamento de carga na rede. Este algoritmo estabelece que os nós com baixa energia devem ser usados com menor freqüência no roteamento e para isso realiza um procedimento local de eleição. A eleição é usada para escolher o nó com mais baixa energia de todos os nós da rede. O nó eleito ficará mais tempo em repouso (modo *sleep)* que seus vizinhos. Como o protocolo é baseado em TDMA e a eleição é totalmente integrada com o tempo alocado para cada nó *(slot* TDMA), o protocolo não diminui a vazão da rede. O DE-MAC assume sincronização dos quadros TDMA e o método eleição dos nós com mínima energia garante o balanceamento de energia na rede.

4.3.2.6. Protocolo TRAMA

O protocolo TRAMA (*Traffic adaptive Multiple Access*) [Rajendran et al., 2003] é baseado em alocação estática de canal TDMA. É projetado para aplicações dirigidas a eventos com coleta contínua ou periódica de dados em RSSFs. A meta principal deste protocolo é ser eficiente

em energia e o método de acesso ao canal garante que não existirão colisões em comunicações *unicast*, *broadcast* ou *multicast*.

O TRAMA é adaptativo ao tipo de tráfego e emprega um algoritmo distribuído de eleição. Este algoritmo determina qual nó pode transmitir em determinado intervalo de tempo (*time-slot*), e não faz reserva para os nós sem tráfego de envio. O algoritmo de eleição é baseado em informações de tráfego de cada nó e seleciona receptores baseados em escalas anunciadas pelos transmissores. As escalas são obtidas pela troca de informações locais de sua vizinhança de dois *hops* e são transmitidas para especificar quais nós serão os respectivos receptores de seu tráfego em ordem cronológica. O TRAMA alterna entre acessos aleatórios e escalonados para acomodar mudanças de topologia, permitindo adição de nós à rede e tolerância a falhas. Consiste de três componentes:

- NP (*Neighbor Protocol*): responsável pela propagação e atualização de informações sobre seus vizinhos de um *hop*. As atualizações são incrementais e permitem determinar o conjunto de vizinhos que serão adicionados ou removidos.
- SEP (*Schedule Exchange Protocol*): permite que os nós troquem informações e escalas da vizinhança de dois *hops*.
- AEA (*Adaptive Election Algorithm*): utiliza as informações da vizinhança e de suas escalas para selecionar transmissores e receptores para o intervalo de tempo atual, enquanto os outros nós selecionam o modo de repouso (*sleep*).

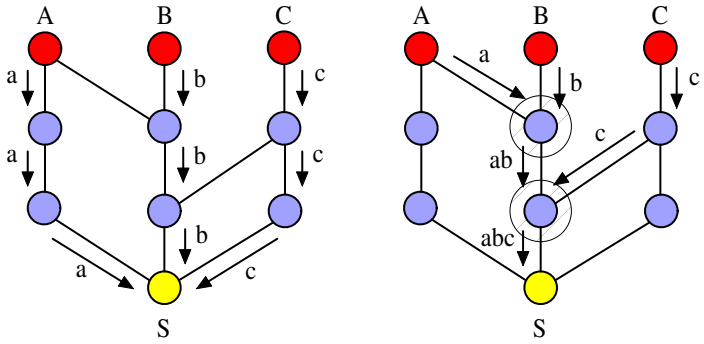
Apesar da troca de informações entre vizinhos tentar criar uma visão global da rede e o protocolo assumir sincronização de quadros TDMA, o TRAMA mostra-se adequado para aplicações insensíveis à latência que necessitam de eficiência em energia e alta taxa de entrega.

4.3.3. Camada de Rede

A principal função da camada de rede é prover o serviço de roteamento que pode ser definido como o processo pelo qual a rede consegue identificar o destinatário das mensagens e encontrar um caminho entre a origem e o destino desta mensagem. Este processo é de fundamental importância em todas as redes de computadores, e em RSSFs não é diferente. Existem diversas formas diferentes de se fazer o roteamento entre os nós em RSSFs, e a eficiência da RSSF será dada, em grande parte, pela forma como o roteamento das mensagens ocorre nesta rede.

As RSSFs assemelham-se às MANETs no sentido de que em geral ambas oferecem um serviço de comunicação *multi-hop*. Contudo, o tipo de aplicação destas redes e os requisitos de roteamento diferem em alguns aspectos. Primeiro, a forma de comunicação típica de uma RSSF é unidirecional no sentido dos nós fontes para o ponto de acesso, como um *multicast* invertido. Segundo, os dados dos nós fontes em geral referem-se a um fenômeno comum, portanto, existe a probabilidade de redundância dos dados transmitidos. Terceiro, normalmente os nós sensores possuirão pouca ou nenhuma mobilidade. Finalmente, a principal restrição nas RSSFs é a limitação de energia. Esta limitação é bem mais crítica nas RSSFs pois os dispositivos são menores e elas supostamente funcionarão em um modo não assistido que não prevê a recarga ou troca das baterias.

Neste contexto de severas limitações, a fusão/agregação de dados tem sido apontada como uma opção que permite otimizar a operação das RSSFs [Heidemann et al., 2001, Intanagonwiwat et al., 2000]. A idéia é pré-processar os dados dentro da rede reduzindo a ocorrência de redundâncias e o número de transmissões para economizar energia. Este paradigma modifica o foco da abordagem tradicional, centrada em endereço, para uma abordagem nova, centrada em dados, que permite a consolidação/sumarização de dados redundantes.



(a) Roteamento centrado em endereços.

(b) Roteamento centrado em dados.

Figura 4.5: Solução centrada em endereço e centrada em dados utilizando fusão de dados.

Para ilustrar o funcionamento do roteamento centrado em dados considere a figura 4.5 que compara o roteamento centrado em endereços com o centrado em dados. Nesta figura, os nós A, B e C enviam dados para o nó sorvedouro S. No roteamento centrado em endereço a difusão destes dados gera 9 mensagens. Na solução centrada em dados o número de mensagens cai para 6. Os nós destacados fazem a fusão dos dados. O primeiro funde as mensagens a e b em ab e o segundo funde as mensagens ab e c em abc. Claramente o paradigma centrado em dados afeta a forma em que os dados são roteados. Contudo, a utilização deste paradigma na forma de endereçamento também pode trazer benefícios conforme é mostrado a seguir.

4.3.3.1. Endereçamento em RSSFs

Um dos propósitos do endereçamento em redes tradicionais é o provimento de informações topológicas para auxiliar a procura por rotas. Uma analogia simples consiste em dizer que endereços funcionam como nomes para pontos de destino específicos (por exemplo, “gostaria de contactar o nó A”). Assim, uma propriedade importante para as redes tradicionais é o endereçamento global único para permitir a identificação de qualquer nó que se deseja estabelecer comunicação. Entretanto, este tipo de endereçamento exige um espaço (bits) suficiente para identificar cada um dos nós da rede. Assim, quanto maior o número de nós maior deverá ser o espaço necessário para seus endereços. Embora isto não represente um fator crítico para as redes tradicionais, nas RSSFs cada bit transmitido reduz o tempo de vida da rede [Pottie and Kaiser, 2000]. Todos os componentes de uma RSSF devem minimizar o consumo de energia para prolongar o tempo de vida da rede. Além disso, o número de elementos em uma RSSF pode ser da ordem de milhares. Assim, o endereçamento tradicional como o

IPv6, tende a ser muito grande aumentando os custos de comunicação. Outro fato importante é que tipicamente as aplicações de RSSFs não estão interessadas no identificador de um nó individual, consultas são feitas com o objetivo de extrair dados de uma região e não de um nó. Conseqüentemente, se faz necessário encontrar novas soluções de endereçamento que atendam as restrições das RSSFs considerando suas particularidades. Nesta seção são discutidas algumas alternativas como: endereçamento espacial, baseado em atributos e de transações.

Endereçamento Espacial. Tipicamente as aplicações de RSSFs não estão interessadas no identificador de um nó individual, consultas são feitas com o objetivo de extrair dados de uma região e qualquer nó capaz de detectar um determinado evento pode responder a esta consulta. Nestes casos, onde é desejável o conhecimento da localidade, o endereçamento espacial (coordenadas geográficas) torna-se interessante [Estrin et al., 1999]. Algoritmos de roteamento que utilizam este tipo de endereçamento são chamados de algoritmos geográficos e serão explorados posteriormente.

O endereçamento espacial global permite que sejam referenciados nós individuais ou grupos de nós através de sua localização geográfica. Entretanto, o tamanho do endereço (codificação das coordenadas geográficas) depende de fatores como a granularidade (precisão) da localização, do tamanho da região a ser monitorada e da quantidade de nós a serem endereçados. Estes fatores podem dificultar a escalabilidade deste esquema tornando o endereço muito grande em relação aos dados que estão sendo transmitidos.

Endereçamento Baseado em Atributos. A idéia de nomeação de dados para RSSFs tem origem em algoritmos como o pioneiro SPIN (*Sensor Protocols for Information via Negotiation*) [Heinzelman et al., 1999], onde os dados são identificados por metadados (descritores). Entretanto, o SPIN assume um endereçamento (global) dos nós sensores.

Na classe de endereçamento baseado em atributos, a comunicação baseia-se em atributos externos à topologia e relevantes para a aplicação, diferentemente do endereçamento espacial que considera a região geográfica dos nós para endereçá-los. Esta forma de endereçamento utiliza a nomeação de dados mudando o foco do endereçamento dos nós (e sua localização) para os dados em si. Neste esquema, os atributos, utilizados para descrever ou nomear um dado, são identificados por chaves únicas que são distribuídas por uma unidade central. A solução proposta utiliza o protocolo de Difusão Direcionada [Intanagonwiwat et al., 2000] (ver seção 4.3.3.1), regras de casamento de padrão e filtros para viabilizar a comunicação e disseminação de dados.

Endereçamento de Transações. Em [Elson and Estrin, 2001] é proposto e avaliado o esquema de endereçamento para RSSFs chamado *Random, Ephemeral TRansaction Identifiers* (RETRI). Ao contrário do esquema tradicional onde são atribuídos identificadores estáticos (como endereçamento de nós ou nomeação de dados), no RETRI os nós selecionam identificadores probabilisticamente únicos, para cada transação nova. Neste esquema, uma transação é definida como qualquer computação na qual é necessário manter algum estado entre os nós

envolvidos.

No RETRI, quando é necessário um identificador único, é gerado um identificador aleatório probabilisticamente único. Entretanto existe a chance de que dois nós utilizem o mesmo identificador ao mesmo tempo. Neste caso ocorre uma colisão de identificadores resultando na perda da transação que deve ser tratada como qualquer outra perda. Neste esquema de endereçamento, para cada pacote é atribuído um identificador aleatório. Uma vez que um identificador é selecionado para o pacote, todos os seus fragmentos recebem o mesmo identificador, permitindo a sua reconstrução do lado receptor. A cada novo pacote é atribuído um novo identificador aleatório. Neste caso uma transação é considerada a transmissão de todo o conteúdo (fragmentos) de um pacote.

O RETRI torna-se interessante para casos onde o tamanho do dado é pequeno em relação ao tamanho de um identificador, e o número de transações de um nó individual é pequeno em relação ao número de nós presentes na rede. O tamanho dos identificadores cresce com a densidade da rede e não com o tamanho total da rede, permitindo a escalabilidade do esquema proposto pelo RETRI.

4.3.3.2. Roteamento Plano

No roteamento plano todos os nós são considerados iguais do ponto de vista funcional, ou seja, a atividade de roteamento é tratada de forma idêntica por todos os nós da rede. Alguns representantes importantes desta classe de algoritmos são apresentados a seguir.

Protocolo Difusão Direcionada. Em [Intanagonwiwat et al., 2000] é proposto um algoritmo chamado Difusão Direcionada. A meta deste algoritmo é estabelecer canais de comunicação eficiente entre os nós sensores e a estação base. Este algoritmo, introduz dois novos conceitos: roteamento baseado nos dados e a agregação de dados. O roteamento baseado nos dados ocorre através da requisição de informação de interesse. Quando algum nó possui alguma informação que é de interesse de outro nó, ele envia esta informação ao nó que requisitou tal informação. O outro conceito, agregação de dados, significa que nós intermediários podem agregar seus dados em um simples pacote para reduzir transmissões e o volume total de dados transmitidos.

O modo de funcionamento básico da difusão direcionada pode ser descrito da seguinte maneira: uma requisição de sensoriamento é disseminada pela rede na forma de um interesse. Essa disseminação configura o gradiente para responder à requisição quanto aos eventos de interesse. O gradiente contém informações sobre a taxa de transmissão e tempo de vida do interesse. Os nós que originam os eventos de interesse iniciam a propagação dos interesses através de caminhos múltiplos. Os nós que originam os interesses reforçam um, ou um pequeno número, de caminhos pelos quais os dados deverão ser entregues. Os interesses são periodicamente atualizados pela estação base.

Protocolo SPIN. O SPIN (*Sensor Protocols for Information via Negotiation*) [Heinzelman et al., 1999] é um protocolo de roteamento para RSSF que usa informações sobre a quantidade de energia disponível em cada sensor para fazer o roteamento. Ele utiliza-se de protocolos de negociação para disseminar as informações de um nó sensor para todos os nós sensores da rede. No SPIN quando um nó percebe que sua energia está perto de um limite pré-estabelecido ele se adapta participando menos da disseminação de dados.

O SPIN funciona em três estágios: ADVertise, REQuest, DATA. O protocolo é iniciado quando um nó obtém novos dados que ele deseja disseminar. Quando o nó possuir dados para compartilhar, ele pode advertir este fato transmitindo uma mensagem ADV contendo metadados (estágio ADV) para seus vizinhos. Ao receber um ADV, os vizinhos do nó verificam se possuem os dados requeridos ou se já requisitaram tal dado. Se não possuem, ele envia ao nó que disseminou a informação uma mensagem de requisição de dados (estágio REQ). O nó que possui o dado a ser transmitido responde à requisição com uma mensagem de dados (estágio DATA). Após receber o dado, o nó vizinho envia uma mensagem ADV a todos os seus vizinhos, informando que ele possui um dado novo e que ele quer repassá-lo. Assim, o ciclo se reinicia.

Protocolo SAR. O protocolo SAR (*Sequential Assignment Routing*) [Sohrabi et al., 2000] visa facilitar o roteamento *multi-hop*. O objetivo deste algoritmo é minimizar a média ponderada de métricas de qualidade de serviço(QoS - *Quality of Service*) através do tempo de vida da rede. Ele leva em consideração os recursos de energia e as métricas de QoS de cada caminho e a prioridade dos pacotes. A seleção do caminho é feita pelo nó que gera o pacote a não ser que a topologia mude o caminho fazendo com que o pacote tenha que ser desviado. Para cada pacote roteado pela rede é computado um peso de medida de QoS como o produto da métrica de QoS e a média da prioridade dos pacotes. A idéia é prover cada pacote com um coeficiente de QoS relativo a sua prioridade.

Protocolo Multi. O Multi [Figueiredo et al., 2004], Protocolo Adaptativo Híbrido para Disseminação de Dados em RSSFs, consiste de uma nova abordagem na construção de algoritmos de disseminação de dados em RSSFs. Trata-se de um protocolo adaptativo para disseminação de dados em RSSFs que reúne características de outros dois protocolos também definidos [Figueiredo et al., 2004]: o SID (*Source-Initiated Dissemination*), um algoritmo reativo onde o processo de disseminação é iniciado a partir da origem dos dados, e o EF-Tree (*Earliest-First Tree*), um algoritmo que constrói e mantém pró-ativamente uma árvore para a disseminação de dados para toda a rede.

A idéia básica do Multi é explorar cenários onde o comportamento da rede pode variar muito. Por exemplo, uma aplicação orientada a eventos pode ter intervalos de tempo longos com baixa ou nenhuma incidência de eventos, mas em determinado instante pode ocorrer uma avalanche de eventos, provocando alto tráfego de dados. Nesse caso, pode-se ter um algoritmo mais adequado para cada instante da rede, podendo ser inviável, ou até mesmo impossível, uma entidade externa agir dinamicamente nessa rede modificando seu comportamento. Assim, a proposta do Multi consiste em adaptar seu funcionamento de forma autônoma adotando o

comportamento de um dos algoritmos que o compõe quando for mais interessante sob a ótica do consumo de recursos da rede, principalmente de energia.

Para realizar a adaptação conforme variação de tráfego na rede, o Multi estabelece um limiar a partir do qual, observado a elevação do tráfego em determinado intervalo de tempo, o esquema de disseminação é alternado. Inicialmente o Multi adota o comportamento do SID, que mostrou-se mais adequado ao tráfego eventual e restrito a poucas fontes de dados, devido ao seu comportamento reativo. Esse algoritmo funciona enviando dados em *broadcast* na rede até que um caminho seja estabelecido pelo nó sorvedouro através do envio de mensagens de requisição a vizinhos específicos, no caso os que levam à entrega de dados de forma mais rápida, formando um caminho reverso à fonte dos dados pelo qual novos dados passarão a ser entregues. Caso o tráfego causado pela elevação do número de fontes ultrapasse o limiar pré-determinado, o comportamento do EF-Tree é assumido, onde o nó sorvedouro passa a agir pró-ativamente construindo e mantendo uma árvore de disseminação para toda a rede. A árvore é construída a partir do *broadcast* de mensagens de controle pelo nó sorvedouro e o ancestral de um nó é tomado quando a primeira mensagem de controle é recebida. Essa adaptação mostrou-se vantajosa pois ao verificar-se a tendência no aumento do número de fontes, uma infra-estrutura de disseminação é criada antecipadamente, evitando os *broadcasts* individuais provenientes do esquema do SID. Ao se ter o tráfego reduzido abaixo do limiar, observa-se uma tendência de diminuição no número de fontes, assim o Multi adapta-se novamente retornando a funcionar como SID.

Protocolo STORM/AD. Em [Nakamura et al., 2004] é proposta uma solução de disseminação em RSSFs composta por um algoritmo de superimposição (*Adaptive Diffusion*) [Bougé and Frances, 1988] que opera sobre a topologia criada e mantida por um algoritmo de auto-organização (STORM – *Self-organizing TOpology discoveRy and Maintenance/Adaptive Diffusion*). O modo de operação do par STORM/AD é mostrado na figura 4.6 onde o algoritmo de auto-organização é executado de forma contínua no plano de substrado enquanto o algoritmo de roteamento executa no plano de superimposição sobre a infra-estrutura disponível.

O STORM é um algoritmo distribuído para descoberta e manutenção de topologia para RSSFs provendo a infra-estrutura necessária para a atividade de disseminação de dados. A topologia resultante deste algoritmo é um grafo acíclico direcionado (com múltiplos caminhos entre cada nó fonte e o sorvedouro) onde o fluxo de dados segue dos nós fontes para o nó sorvedouro. Como a topologia disponibilizada pelo STORM é um grafo acíclico direcionado, o *Adaptive Diffusion* pode escolher qualquer caminho sem se preocupar com a detecção de ciclos. Assim, quando um nó precisa enviar dados, ele avalia seus pais (levando em conta métricas como menor caminho e caminho de maior energia) escolhendo para qual será enviado o pacote de dados.

O diagrama esquemático do algoritmo é mostrado na figura 4.7. Primeiro, o nó calcula um *coeficiente de adaptação* para cada um de seus pais. A seguir, todos os coeficientes são avaliados e o pai que (supostamente) levar para o melhor caminho em direção ao sorvedouro

(aquele com melhor coeficiente de adaptação) é escolhido. Os parâmetros adequados para o cálculo do coeficiente de adaptação e a função de avaliação devem ser escolhidos de acordo com os requisitos de cada aplicação considerando métricas como: menor caminho, caminho de maior energia, maior economia (agregação), melhor relação sinal/ruído e/ou melhor distribuição de consumo. Neste esquema de roteamento, a agregação de dados ocorre sempre que duas rotas se sobrepõem.

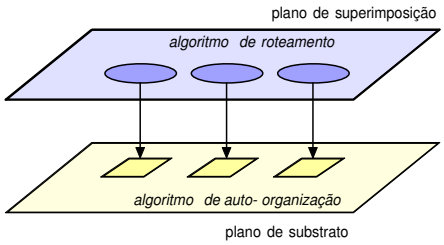


Figura 4.6: Auto-organização e interação de roteamento.

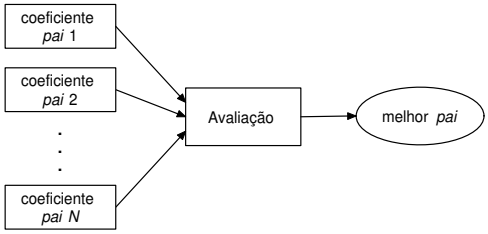


Figura 4.7: Funcionamento do algoritmo do *Adaptive Diffusion*.

Protocolo TinyOS Beaconing. O TinyOS Beaconing [Beaconing, 2004] é o protocolo de roteamento utilizado nos nós sensores da plataforma Mica Motes da Universidade de Berkeley, e tem como requisito o funcionamento em redes com hardware restrito. O protocolo constrói periodicamente uma árvore de caminho mínimo (*Minimum Spanning Tree*) a partir da Estação Base. A Estação Base propaga uma mensagem, chamada de *beacon*, que é propagada na rede com o intuito de criar a árvore de roteamento. Por se tratar de um protocolo simples e geral, seu desempenho é inferior a protocolos desenvolvidos para aplicações específicas.

Protocolo PROC. PROC (Proactive ROuting with Coordination) [Macedo et al., 2004] é um protocolo de roteamento desenvolvido para redes de sensores homogêneas e estacionárias, onde nós enviam dados periodicamente para uma Estação Base. O protocolo considera que os nós possuem capacidade restrita de processamento e comunicação, sendo projetado visando uma implementação compacta.

O PROC utiliza o conceito de coordenadores para construir um *backbone* de roteamento, que é uma árvore com raiz na Estação Base. Os nós que não pertencem ao *backbone* conectam-se diretamente a um nó coordenador. O *backbone* é reconstruído periodicamente, para que o consumo dos nós seja balanceado. O processo de definição dos nós que irão compor o *backbone* é composto de duas partes. Na primeira parte, os nós utilizam heurísticas para determinar se serão ou não coordenadores. Na segunda parte, é feita a complementação do *backbone* caso este não tenha sido completamente formado.

4.3.3.3. Roteamento Hierárquico

No roteamento hierárquico são estabelecidas duas classes distintas de nós: nós fontes e líderes de grupo (*cluster heads*). Os nós fontes simplesmente coletam e enviam os dados para o líder de seu grupo que pode executar uma fusão/agregação destes dados antes de enviá-lo para o ponto de acesso. Todos os nós são considerados iguais do ponto de vista funcional. Alguns algoritmos desta classe de algoritmos são apresentados abaixo.

Protocolo LEACH. O protocolo LEACH (*Low-Energy Adaptive Clustering Hierarchy*) [Heinzelman et al., 2000] tem por objetivo reduzir o consumo de energia. O protocolo foi desenvolvido para redes homogêneas (ver tabela 4.1) e utiliza ciclos durante os quais são formados agrupamentos de nós, denominados *clusters*, onde um nó é escolhido como líder. No LEACH todos os nós da rede iniciam um ciclo ao mesmo tempo, mas não é especificado como obter este grau de sincronização na rede. Para uma rede que está em atividade durante um longo período, o escorregamento do relógio pode fazer com que os nós comecem um novo ciclo em momentos inoportunos. O líder do *cluster* é responsável por repassar os dados do seu *cluster* para a estação base com um único *hop*, o que limita o tamanho da rede em função do raio de alcance do rádio.

Protocolo LEACH-C. O LEACH-C [Lindsey et al., 2002] é uma variação do LEACH [Heinzelman et al., 2000] que centraliza as decisões de formação dos grupos na estação base. A maior vantagem desta abordagem centralizada é a criação e distribuição mais eficiente de grupos, na rede. Cada nó, na fase de inicialização da rede, envia sua posição geográfica e energia disponível para a estação base. Baseando-se nesta informação, a estação através de processos de *simulated annealing*, determina os grupos de forma centralizada. Quando os grupos e seus líderes são determinados a estação base envia uma mensagem que contém o identificador do líder para cada nó. Após esta fase, os nós agem como no LEACH original comunicando-se apenas com seu líder.

Protocolo TEEN. O TEEN (*Threshold sensitive Energy Efficient sensor Network*) [Manjeshwar and Agrawal, 2001] é um algoritmo de roteamento hierárquico similar ao LEACH exceto pelo fato de que os nós sensores podem não possuir dados a serem transmitidos de tempos em tempos. Os autores deste protocolo propõem classificar as redes de sensores em redes pró-ativas e redes reativas. Uma rede pró-ativa monitora o ambiente continuamente e possui dados a serem enviados a uma taxa constante. Em uma rede reativa os nós somente enviam dados quando a variável sendo monitorada se incrementa acima de um certo limite. TEEN utiliza a estratégia de formação de líder do LEACH, mas adota uma estratégia diferente na fase de transmissão de dados. Ele faz o uso de dois parâmetros chamados *Hard Threshold* (Ht) e *Soft Threshold* (St) para determinar a necessidade de transmissão do dado coletado. Se o valor exceder *Ht* pela primeira vez, ele é armazenado em uma variável e transmitido durante o intervalo (*slot*) de tempo alocado a transmissão do nó. Em seguida, se o valor monitorado

exceder o valor armazenado por uma magnitude de *St* o nó transmite o dado imediatamente. O valor enviado é armazenado para comparações futuras.

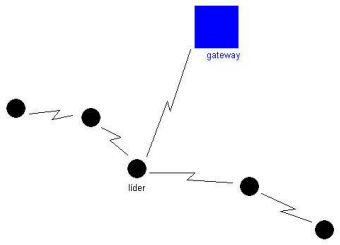


Figura 4.8: Funcionamento do PEGASIS.

Protocolo PEGASIS. O PEGASIS (*Power-Efficient Gathering in Sensor Information Systems*) [Lindsey and Raghavendra, 2002] é um protocolo para RSSF baseado no conceito de correntes. Cada nó troca informações apenas com os vizinhos mais próximos formando uma corrente entre os nós, e apenas um nó é escolhido a cada momento para transferir as informações coletadas ao nó *gateway* (ver figura 4.8).

Portanto, o número de trocas de mensagens será baixo e a comunicação será realizada entre nós próximos uns dos outros. Espera-se com isso que a energia gasta seja menor, se comparada a outros protocolos que requerem muitas trocas de mensagens para eleger líderes e formar grupos, e protocolos em que os nós constantemente trocam mensagens com o nó *gateway* de forma direta (o *gateway* geralmente se encontra distante dos nós). Isto implica um tempo de vida maior para cada nó e um consumo menor da largura de banda da rede. O PEGASIS assume o seguinte:

- O nó *gateway* (estação base) situa-se estacionado à uma distância fixa da rede;
- Os nós são capazes de transmitir dados diretamente para o nó *gateway* e para qualquer outro nó;
- Cada nó possui informação de localização dos outros nós;
- Os nós são homogêneos e com o nível de energia uniforme;
- Os nós não são móveis. A cada round um nó é escolhido para transmitir a informação à estação base.

Protocolo ICA. O protocolo ICA (*Inter Cluster Routing Algorithm*) [Habib et al., 2004] é baseado no LEACH, sendo idealizado para aumentar o tempo de vida e o número de pacotes enviados na rede. O ICA inicia com a estação rádio base enviando um *broadcast* para todos os nós informando sua posição geográfica. Após esta fase, os nós sabem a posição geográfica da estação base e é assumido que também sabem suas próprias posições. No ICA os nós são agrupados em *clusters* que seguem as mesmas regras de formação do LEACH, a não ser pela decisão de qual *cluster* os nós vão participar. Esta informação é dada pela proximidade dos nós aos *cluster heads*. O nó vai estar ligado sempre ao *cluster head* mais próximo. O processo de formação de *clusters* dissemina a informação da formação de *clusters* pelos *clusters* vizinhos. No ICA, ao contrário do LEACH, os *cluster heads* tentam não enviar as mensagens diretamente para a estação base. Ao invés disto eles, em uma abordagem gulosa, enviam as mensagens para o *cluster head* mais próximo, na direção da estação base. O objetivo é economizar energia enviando as mensagens ponto a ponto para nós que estão a uma distância menor da estação base.

Desta forma a quantidade de energia consumida por cada nó da rede diminui e a quantidade de energia total da rede aumenta. Para evitar o problema da morte prematura dos nós perto da estação base, os *cluster heads* no ICA podem se recusar a retransmitir mensagens de outros *clusters* para a estação base. Isto ocorre quando o *cluster head* percebe que está ficando sem energia. Para evitar que não possa enviar as mensagens do seu próprio *cluster* ele para de rotear mensagens de outros *clusters* para a estação base. Quando ocorre uma recusa em retransmitir dados, o *cluster head* que requisitou o serviço de roteamento envia a mensagem diretamente a estação base, da mesma forma como ocorre no LEACH.

Esta abordagem tenta impedir o aparecimento de áreas descobertas perto da estação base. Isto deveria ocorrer rapidamente uma vez que todas as mensagens da rede teriam que passar por estes nós antes de chegar a estação base.

4.3.3.4. Roteamento Geográfico

O roteamento geográfico utiliza informações geográficas para rotear seus dados. Estas informações costumam incluir a localização dos nós vizinhos. Os dados de localização podem ser definidos a partir de um sistema de coordenadas globais (GPS - *Global Position System*) ou mesmo de um sistema local válido somente para os nós da rede ou válidos somente para subconjuntos de nós vizinhos. Os principais algoritmos geográficos utilizados em RSSFs são apresentados a seguir.

Geographic Routing without Location Information. O *Geographic Routing without Location Information* [Rao et al., 2003] é um algoritmo que visa atribuir coordenadas virtuais aos nós. Assim, os nós não precisam necessariamente saber a sua coordenada real. Apesar de assumir que os nós sensores sabem onde estão, a localização dos nós não é condição necessária para o funcionamento do protocolo. O autor observa três cenários onde os nós podem possuir coordenadas virtuais:

1. *Os nós da borda da rede sabem a sua localização.* É possível determinar as coordenadas dos nós restantes a partir da posição dos nós de borda.
2. *Os nós da borda da rede sabem que estão na borda da rede, mas não sabem a sua localização.* Os nós da borda enviam para toda a rede uma mensagem de *HELLO*, para que possam determinar a sua distância em relação aos outros nós que estão na borda. Assim, todos os nós da borda descobrem as coordenadas de todos os outros nós da borda utilizando triangulação. Em seguida, os nós que não estão na borda utilizam o método descrito no primeiro cenário para descobrir as suas coordenadas.
3. *Nós não sabem se estão na borda, nem a sua localização.* Neste caso é adicionado um passo inicial ao cenário anterior. Uma mensagem de *HELLO* é enviada a toda rede, com o intuito de identificar os nós que estão na borda da rede. Caso um nó está em uma maior distância da mensagem de *HELLO* que todos os nós que estão a uma distância de até dois *hops* do nó, então este é um nó de borda. Após a determinação dos nós de borda, é utilizado o método descrito no cenário anterior para determinar as coordenadas dos nós restantes.

Após determinar a sua coordenada virtual, os nós de borda realizam o roteamento segundo as seguintes regras:

- O pacote é roteado para o nó mais próximo em direção ao destino;
- Se não existe nenhum nó mais próximo do destino do que o nó atual, é verificado se o pacote é destinado a este nó. Caso seja, o pacote chegou ao seu destino. Caso não seja, não é possível entregar o pacote;

GeoMote. O GeoMote (*Geographic Multicast for networked sensors*) [Broadwell et al., 2004], é baseado no GeoCast [Navas and Imielinski, 1997] (protocolo desenvolvido para redes "Internet-like"). Os destinatários das mensagens são endereçados através de polígonos. Desta forma é possível realizar comunicações *multicast* localizadas. No GeoMote, cada nó possui uma função específica durante todo o tempo de vida da rede, definida no momento da sua programação. Existem três categorias de nós sensores: GeoHosts (que produzem dados), GeoRouters (que repassam dados produzidos pelos GeoHosts) e os GeoGateways (que atuam como pontos de entrada e saída de dados).

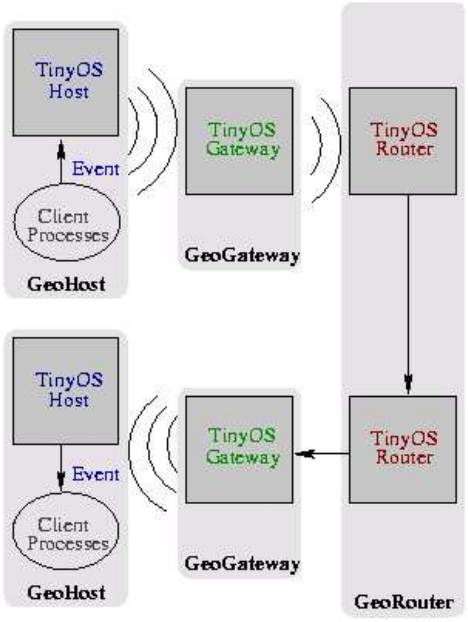


Figura 4.9: Envio de dados no GeoMote.

Para transmitir seus dados, o GeoHost que deseja comunicar com outro nó da rede repassa pacotes para um GeoGateway na sua vizinhança. O GeoGateway irá repassar os dados para um GeoRouter, que por sua vez encaminha os dados até o GeoGateway mais próximo do nó (ou região) alvo da comunicação através de um caminho *multi-hop* (figura 4.9). O caminho de propagação das mensagens é definido por um algoritmo guloso. Neste algoritmo, os GeoRouters repassam os pacotes para o GeoRouter mais próximo dos destinatários da mensagem. Por definir funções estaticamente aos nós, o protocolo é inadequado para redes onde os nós são lançados ou posicionados de forma arbitrária. Os GeoHosts, por exemplo, devem possuir pelo menos um GeoGateway ao alcance do seu rádio, o que pode não acontecer caso os nós sejam posicionados arbitrariamente.

GEAR. O GEAR (*Geographical and Energy Aware Routing*) [Yu et al., 2001] é um protocolo de roteamento geográfico que procura minimizar o consumo de energia da rede. Como o GeoMote, são endereçadas regiões da rede através de retângulos. O repasse de dados utiliza um algoritmo guloso, onde o nó que irá repassar os dados é aquele que possui o menor custo de envio até a região desejada. O custo de envio é calculado em função da distância e energia residual dos nós que compõem a menor rota até a região especificada. Inicialmente, a função custo é aproximada. A cada pacote enviado para a região, a função custo

é recalculada, de forma a otimizar o caminho de repasse dos dados. Ao encontrar a região destinatária dos dados, o protocolo difunde os pacotes através de uma partição recursiva da região em quatro seções. O pacote é enviado para um nó de cada uma das seções, e o algoritmo é aplicado recursivamente, até que as subseções sejam vazias. Em regiões onde a densidade dos nós é pequena, a difusão dos dados é feita via *broadcast*.

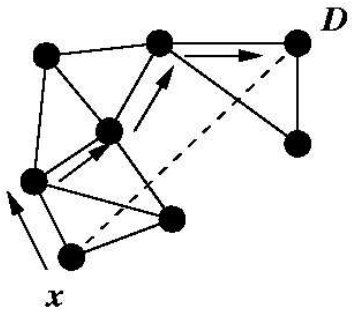


Figura 4.10: GPSR.

O protocolo GEAR se destaca dos demais algoritmos geográficos encontrados na literatura por utilizar informações de toda a rota até o destinatário. O uso de informações de nós distantes permite uma rota mais eficiente, ao custo de um maior tempo de convergência. Em redes onde há mobilidade de nós, o protocolo irá prover rotas menos eficientes que aquelas encontradas em cenários fixos. Além disto, existem vários casos críticos que necessitam de mecanismos específicos para seu tratamento, o que aumenta a complexidade do protocolo.

GPSR. Ao contrário dos protocolos anteriores, o GPSR (*Greedy Perimeter Stateless Routing*) permite endereçar apenas um nó [Karp and Kung, 2000]. O GPSR utiliza dois algoritmos para rotear dados. Quando um nó identifica um vizinho que está mais próximo do destino, o protocolo repassa os dados para este vizinho. Se não existe um vizinho mais próximo, o pacote deve ser repassado para um nó mais distante, para evitar uma região onde a cobertura de nós é baixa. Nestas situações, o protocolo usa o algoritmo de roteamento de perímetro*Perimeter Routing* que constrói um grafo planar para identificar para qual vizinho repassar os dados. Uma vez montado o grafo, a regra da mão direita é utilizada para determinar o próximo *hop* da comunicação, conforme mostrado na figura 4.10, onde o nó x repassa o pacote para o seu vizinho à direita da semi-reta $\overline{xD}$, até encontrar um nó que esteja mais próximo de D em relação a x. Ao determinar que a distância do pacote até o seu destinatário volta a diminuir o protocolo volta a rotear os dados utilizando a abordagem gulosa.

Como vantagens do protocolo podemos destacar o uso de informações locais da vizinhança para roteamento e o uso de algoritmos geométricos simples, que possibilitam a implementação do protocolo em nós sensores com poucos recursos de memória e processador. O protocolo assume que é possível identificar todos os nós da rede eficientemente via informações geográficas. Para facilitar a construção desta tabela, os nós da rede operam em modo promíscuo, armazenando as informações de localização contidas nos pacotes interceptados. Com esta abordagem, os autores argumentam que a atualização dos dados geográficos é facilitada. Em contrapartida, o rádio deve sempre permanecer ligado, o que consome mais energia.

4.3.4. Protocolos de Transporte

Ao contrário das redes tradicionais, o uso de protocolos de transporte em RSSFs nem sempre é necessário. A maioria das aplicações de RSSFs admitem a perda de dados, assim um mecanismo

elaborado para garantia de envio de dados não é justificado. Vários protocolos de roteamento utilizam técnicas com o intuito de diminuir a perda de dados, como o Difusão Direcionada, que repassa os dados através de vários caminhos. Apesar disso, algumas aplicações ou tarefas na rede necessitam de entrega confiável de dados. Podemos citar como serviços de tal tipo a reprogramação de nós e funções de gerenciamento, como desligamento de nós.

4.3.4.1. Protocolo PSFQ

O PSFQ (*Pump Slowly, Fetch Quickly*) [Wan et al., 2002] é um protocolo de transporte desenvolvido tendo em vista a adaptabilidade a diferentes condições de rede. O protocolo trabalha com correção local de erros, utilizando pra isto confirmação ponto a ponto. Segundo os autores, a confirmação ponto a ponto é mais escalável e robusta em ambientes com altas taxas de erros, como as RSSFs. O PSFQ é adaptativo, ou seja, caso a rede apresente um percentual de falhas baixo, o envio irá se assemelhar ao repasse tradicional de dados. Em casos de falhas freqüentes, o protocolo tem um comportamento *store and forward*.

Com o objetivo de identificar perdas de dados, o emissor transmite a mensagem em fragmentos numerados. Cada fragmento recebido é mantido em *cache* em cada nó intermediário da comunicação. Esta *cache* é utilizada para identificar quais fragmentos ou seqüencias de fragmentos (janelas) foram perdidos.

O PSFQ trabalha com 3 tipos de operação: *push*, *fetch* e *report*. As operações de *push* são utilizadas para enviar fragmentos de uma mensagem para o próximo *hop* no caminho até o destinatário. A operação de *fetch* é utilizada quando um nó intermediário identifica que fragmentos não foram recebidos, e tem como objetivo requisitar a retransmissão de fragmentos perdidos. A operação de *report* é utilizada pelos nós receptores da mensagem para notificar o emissor que o recebimento foi completado. A operação de *push* consiste em um repasse periódico de fragmentos de uma mensagem. Os fragmentos são repassados em seqüencia, em intervalos regulares. Caso um nó receba um fragmento com número de seqüencia maior que o esperado, o protocolo identifica que houve um fragmento perdido, e este realiza a operação de *fetch* reativo, para recuperar os fragmentos perdidos. Ao escutar uma mensagem de *fetch*, os nós verificam se possuem um dos fragmentos requisitados em sua *cache*. Caso possuam, irão transmitir o dado para o nó que requisitou o fragmento.

O *fetch* é um NACK (*Negative Acknowledgement*) para uma ou mais janelas de fragmentos. Para garantir que a perda de dados não atrase a comunicação com o próximo *hop*, o nó envia mensagens de *fetch* agressivamente, até que todos os fragmentos perdidos sejam recebidos. O *fetch* também pode ser pró-ativo. Neste caso, o nó envia mensagens de *fetch* caso um fragmento não chegue no tempo esperado. O *fetch* pró-ativo possibilita a recuperação dos fragmentos finais de uma mensagem, pois o *fetch* reativo depende do recebimento de um outro fragmento com número de seqüência maior. Caso um ou mais fragmentos finais sejam perdidos, o *fetch* pró-ativo é utilizado.

4.3.4.2. Protocolo ESRT

O protocolo ESRT (*Event-to-Sink Reliable Transfer*) [Sankarasubramaniam et al., 2003] é um protocolo projetado para RSSFs baseadas em eventos. Segundo os autores, um protocolo de transporte em redes de sensores deve ser utilizado com o objetivo de reconhecer um evento de forma confiável. O reconhecimento é feito na estação base, através do recebimento de mensagens de vários sensores. Neste trabalho, considera-se que o reconhecimento confiável de um evento é feito através do recebimento de um número de mensagens apropriado. Caso o número de mensagens recebidas seja inferior ao necessário, o evento não é reconhecido de forma confiável. Desta forma, o objetivo do ESRT é ajustar a taxa de envio de dados de cada nó para que a taxa de recebimento de pacotes reportando um evento esteja próxima do valor necessário para o reconhecimento confiável do mesmo.

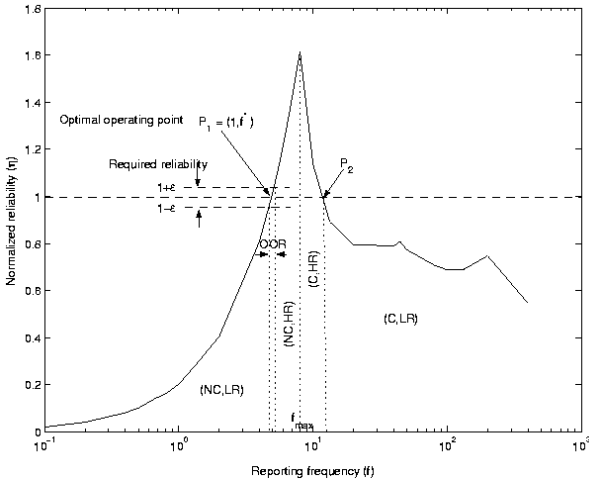


Figura 4.11: ESRT.

O protocolo utiliza o conceito de estados de operação. Cada estado é definido pelo comportamento da rede em termos da confiabilidade dos dados coletados e congestionamento da rede. Para medir o congestionamento dos nós, verifica-se o tamanho da fila de pacotes a serem enviados. Caso esta ultrapasse um tamanho pré-determinado, o nó comunica o fato à estação base através de um cabeçalho do protocolo. Os autores identificaram os comportamentos possíveis da rede e definiram estados de funcionamento. Estes estados influem na confiabilidade dos eventos reportados, como mostra a figura 4.11 que representa a confiabilidade de reconhecimento de um evento em função do período de envio de dados. O eixo Y representa a confiabilidade normalizada de reconhecimento de um evento. O objetivo do protocolo é operar em um estado de funcionamento ótimo, onde a confiabilidade é próxima de 100%, e o envio de mensagens é minimizado. Foram definidos os seguintes estados (representados na figura 4.11):

1. (NC,LR) Intervalo sem congestionamento e baixa confiabilidade.
2. (NC,HR) Intervalo sem congestionamento e alta confiabilidade.
3. (C,HR) Intervalo com alto congestionamento e alta confiabilidade.
4. (C,LR) Intervalo com alto congestionamento e pequena confiabilidade.
5. (OOR) Intervalo de operação ótimo.

A estação base calcula em que estado de operação a rede se encontra em intervalos regulares de tempo. Ao calcular o estado da rede na estação base, o protocolo evita o consumo de energia e processamento nos sensores, o que poderia inviabilizar a operação. Caso a rede se encontre fora do estado de operação ótimo, a taxa de envio de dados dos nós é ajustada de acordo com as regras da tabela 4.7. Esta taxa de envio é repassada para os nós, que a utilizam a

partir daquele momento.

Estado da rede	Ação
(NC,LR)	Aumenta f multiplicativamente de form a encontrar a confiabilidade requerida o mais rápido possível
(NC,HR)	Diminui f de forma conservadora de forma a não afetar a confiabilidade
(C,HR)	Diminui f agressivamente até estado (NC,HR) para evitar congestionamento na rede o mais rápido possível
(C,LR)	Diminui f exponencialmente para evitar congestionamento na rede o mais rápido possível
(OOR)	Mantém f anterior

Tabela 4.7: Ações tomadas pelo protocolo ESRT em cada estado de operação. f denota a freqüência de envio de mensagens pelos nós.

4.3.4.3. Protocolo RMST

O RMST (*Reliable Multi-Segment Transport*) [Stann and Heidemann, 2003] é um protocolo de transporte desenvolvido para operar em conjunto com o Difusão Direcionada (ver seção 4.3.3.1). O protocolo tem como objetivo garantir a entrega de mensagens dos nós sensores até a estação base, como acontece nas redes sem fio usuais. Por se tratar de um protocolo desenvolvido especificamente para o uso com o Difusão Direcionada, o RMST implementa transmissão confiável com um pequeno *overhead*. A dinamicidade da topologia é tratada na camada de rede, desta forma a camada de transporte implementa apenas um sistema de confirmação ponto a ponto. O protocolo utiliza *cache* de dados associadas ao gradiente do Difusão Direcionada para garantir que a recuperação de dados perdidos ocorra localmente. A detecção de fragmentos perdidos é feita através de temporizadores. Caso um fragmento não seja recebido no tempo especificado, é enviado um NACK para os nós que estão no sentido inverso do gradiente, requisitando os fragmentos perdidos. Os nós que possuem um dos fragmentos em *cache* o repassam para o nó. Caso o fragmento não esteja em *cache*, o NACK é repassado em sentido contrário ao gradiente até que o fragmento seja encontrado.

4.4. Arquitetura de Nós Sensores

Nas seções anteriores, foram apresentados diferentes protocolos de comunicação projetados para RSSFs. No projeto de um RSSF, os protocolos de uma camada usaram as funcionalidades dos protocolos de outra camada, e obviamente, não é qualquer protocolo de uma camada que funcionará adequadamente com outro protocolo de uma camada adjacente.

Além disso, cada aplicação possui uma demanda diferente de requisitos dos protocolos de comunicação, o que leva à discussão de como "montar" perfis de pilha de protocolos para determinado tipo de arquitetura de nós sensores. Nesta seção, serão apresentadas algumas

plataformas que vêm sendo propostas ou adotadas na construção de RSSFs, tanto por projetos acadêmicos quanto por fabricantes de dispositivos para esse fim. Devido às restrições existentes nas RSSFs, essas pilhas geralmente são compostas somente pelos protocolos da camada de enlace (MAC) e rede (roteamento), que serão apresentados nos exemplos de plataforma descritos nas próximas seções. É importante salientar que a tecnologia para projetar e construir RSSFs está comercialmente disponível e tende a se tornar cada vez mais acessível com a produção em larga escala de diferentes tipos de micro-sensores [Motes, 2002, JPL, 2002, μAMPS, 2002, Dust, 2002, Pico, 2003, WINS, 2003, Millennial Net, 2004].

A figura 4.12 apresenta alguns exemplos de nós sensores sem fio resultantes de pesquisas em diversas instituições, como o COTS Dust e o Smart Dust [Dust, 2002] da *Universidade da California, Berkeley*, WINS [WINS, 2003] (*Wireless Integrated Network Sensors*) da *Universidade da California, Los Angeles* e JPL Sensor Webs [JPL, 2002] do *Jet Propulsion Lab* da NASA.

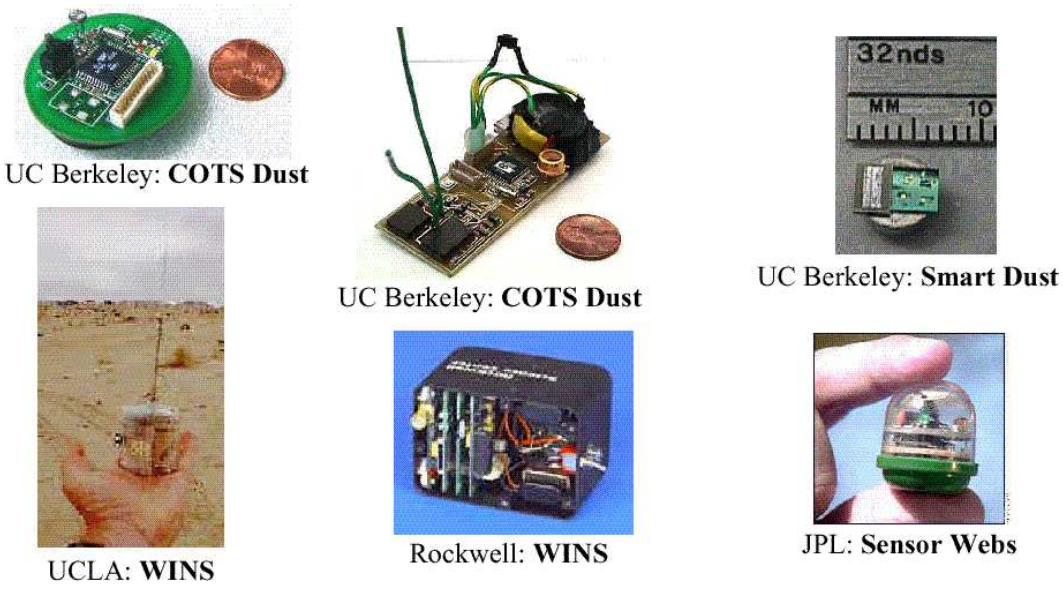


Figura 4.12: Projetos acadêmicos de nós sensores.

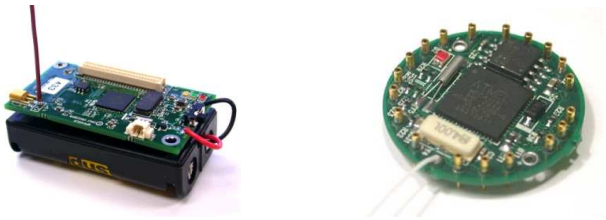
4.4.1. Projeto Macro Motes (COTS Dust)

Os pesquisadores da Universidade de Berkeley desenvolveram nós sensores conhecidos como Motes. Um dos principais objetivos do projeto desses dispositivos é o baixo consumo de energia. Os nós sensores Motes podem ser encontrados sob diferentes versões, tamanhos e características. A primeira geração, implementada como projeto de tese [Hill, 2000] de Seth Hollar em 2000, é conhecida como Macro Motes ou COTS Dust Mote. Existem algumas variações de projeto Macro Motes que são conhecidas como WeC Mote, RF Mote, Laser Mote, CCR Mote, Mini Motes, MALT Motes e IrDA Motes. O transceptor RF é o TR 1000 que opera em freqüência 916,5 MHz, com capacidade de transmitir em média 10 kbps. O sistema operacional

deste nós é o TinyOS (ver seção 4.5, que é dirigido a eventos e ocupa apenas 178 bytes de memória.

Em seguida aos projetos Macro Motes, os pesquisadores projetaram o Rene Motes e finalmente, a última geração de desenvolvimento, formada pelos MICA Motes (ver seção 4.4.2) e Smart Dust (ver seção 4.4.3).

4.4.2. Projeto Mica Motes



(a) Mica2 Motes (b) Mica2 Dot

Figura 4.13: Mica Motes.

Os nós sensores Mica Motes [Motes, 2002] também são desenvolvidos pelos pesquisadores da Universidade de Berkeley. Possuem características diferentes dos Macro Motes (ver seção 4.4.1). A plataforma Mica Motes é comercializada pela Crossbow [Crossbow, 2004] e é uma das mais empregadas em projetos envolvendo RSSFs. A unidade de sensoriamento de cada nó Mica Mote pode ser equipada com uma variedade de sensores, tais como acústico, temperatura, aceleração, luminosidade e pressão. O microcontrolador normalmente incorpora uma unidade de processamento RISC, memória RAM e FLASH, conversores analógico-digitais, temporizadores e controladores de interrupção.

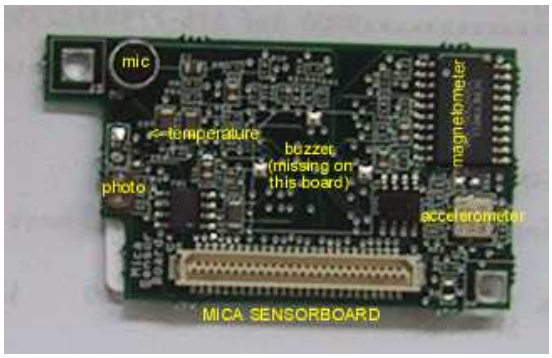


Figura 4.14: Placa de sensores.

O Mica2 Mote (ver figura 4.13(a)) é um nó sensor de baixo consumo de energia (*low-power*), que utiliza o rádio CC1000 (ver seção 4.3.1) para comunicação sem fio e possui um barramento de 51 pinos que permite conexão com uma placa contendo um ou mais sensores. Estas placas são denominadas *sensorboards* (ver figura 4.14). Este nó sensor também possui uma memória FLASH externa de 4 Mbits que serve como memória secundária devido a pequena memória interna do microprocessador ATMEGA 128L da Atmel. Este é um microcontrolador de 8 bits, com 128 Kbyte flash ROM (*Read Only Memory*), 4KB RAM (*Random Access Memory*), ADC (Conversor Analógico Digital) de 10 bits. O Mica2 Mote é alimentado por duas pilhas alcalinas AA (2850mAh). Mica2 Dot (ver figura 4.13(b)) é uma versão menor do Mica2 Mote que possui os mesmos recursos computacionais, com exceção dos componentes da bateria (uma bateria de litium 3B45 (1000mAh) e do barramento (18 pinos).

Ela adota o TinyOS [TinyOS, 2004] (descrito na seção 4.5, também desenvolvido pela

Universidade de Berkeley, como sistema operacional de distribuição, sendo este de código aberto, modular e de fácil personalização. Os protocolos de comunicação da plataforma são disponibilizados como módulos do TinyOS. Na versão 1.1 do sistema, o MAC era CSMA/CA, mesmo esquema do 802.11, o que traz como conseqüência um maior consumo de energia pelos motivos descritos anteriormente. Já na versão 1.1.3, o B-MAC (ver seção 4.3.2.4) foi introduzido. Também baseado no CSMA/CA, ele realiza controle de ganho e filtragem de canal, permite o aumento da largura de banda e a redução a 85% da utilização do canal, reduzindo consumo de energia. Como algoritmo de roteamento se tem a implementação de uma árvore "menor-caminho-primeiro" a partir de um nó raiz (nó sorvedouro) que tem sua confiabilidade aumentada através de um esquema de seleção de pais. Mudanças topológicas são comportadas através de atualizações periódicas, assim como algumas soluções descritas anteriormente. Embora a característica modular da plataforma em questão permita adotar outros protocolos de forma flexível, as soluções existentes atualmente visam atender as restrições impostas pelas RSSFs, porém, não tendem adequadamente aplicações de tráfego eventual ou sob demanda pelos mesmos motivos descritos da plataforma do MANTIS (ver seção 4.4.9).

4.4.3. Projeto Smart Dust

O Projeto Smart Dust [Dust, 2002] é desenvolvido pela Universidade de Berkeley e tem por objetivo reduzir o tamanho dos nós sensores para que estes apresentem as dimensões de um grão de poeira, ou seja, um cubo de aproximadamente um milímetro. Os componentes disponíveis para este dispositivo serão um sensor, uma bateria, um circuito analógico, um dispositivo de comunicação óptica bidirecional e um microprocessador programável.

A comunicação através de transceptores de Rádio Freqüência (RF) é bastante inadequada para os nós deste tipo, devido a vários aspectos. Um deles é o fato de que as antenas seriam muito grandes para os Smart Dust, e outro é o consumo de energia, que seria alto para a disponibilidade do nó. Assim sendo, a transmissão óptica é a mais adequada, e é utilizada tanto na forma passiva quanto ativa [Dust, 2002].

4.4.4. Projeto MicroAmps

Os pesquisadores do Massachusetts Institute of Technology (MIT) são os responsáveis pelo desenvolvimento do μAMPS. Os nós sensores μAMPS (μ-Adptative Multi-domain Power Aware Sensor) [μAMPS, 2002] possuem uma política de gerenciamento de energia, conhecida por *power-aware* ou *energy-aware*, que permite que o nó sensor seja capaz de fazer com que seu consumo de energia se adapte às características e variações do ambiente onde se encontra, dos recursos que ele próprio dispõe e das requisições dos usuários da rede. Esta metodologia é, portanto, ideal para aplicações onde existem muitas variações no ambiente. O nó sensor μAMPS utiliza o rádio transceptor LMX3162 da *National Semiconductor*. Opera na banda ISM na freqüência 2,45 GHz, consegue um alcance entre 10 e 100 metros e taxa de 1 Mbps, em transmissões sem fio, ponto a ponto. A camada de enlace utiliza TDMA e está integrada ao rádio PCB, e age como um bloco de memória de armazenamento.

4.4.5. Projeto WINS

O Rockwell Science Center em colaboração com pesquisadores da Universidade da California, Los Angeles (UCLA), desenvolveram o protótipo de um nó sensor, chamado WINS [WINS, 2003]. O dispositivo combina capacidade de sensoriamento (tais como sísmica, acústica e magnética) com um processador RISC embutido e um rádio de transmissão. O módulo do rádio usa o Conexant RDSSS9M que implementa uma comunicação RF spread spectrum a uma freqüência de 900 MHz (ISM). O rádio opera em um dos 40 canais, escolhido pelo controlador. O alcance do rádio pode ultrapassar os 100 metros. A camada de enlace (MAC) utiliza TDMA permitindo uma taxa de 100 kbps. Os pesquisadores da Rockwell desenvolveram softwares para os protocolos básicos de comunicação, um Kernel *runtime*, *drivers* para os sensores, aplicações para processamento de sinais e APIs.

4.4.6. Projeto JPL

O Jet Propulsion Laboratory (JPL) [JPL, 2002] do California Institute of Technology está desenvolvendo um projeto chamado SensorWeb. Este projeto consiste em um sistema sem fio, com nós sensores que comunicam-se entre si, distribuídos espacialmente, que podem ser dispostos para monitorar e explorar novos ambientes. O laboratório JPL foi formado para atender aos interesses da NASA, que tem como meta a exploração do Sensor Web em diversas aplicações. O alcance do rádio de transmissão RF pode chegar a 40 metros, com uma taxa de transmissão de 20 kbps a uma freqüência de 916 MHz.

4.4.7. Projeto Medusa

Medusa MK-2 [CENS, 2004] é um nó sensor desenvolvido no Laboratório de Engenharia Elétrica da Universidade da Califórnia com objetivo de se fazer testes reais de RSSFs, que operam sem a supervisão humana. O rádio possui uma potência de transmissão de 0,75 mW e seu alcance pode chegar aos 20 metros. A taxa de transferência pode variar de 2,4 kbps até 115 kbps. A comunicação é feita através de um rádio TR1000 e um barramento serial RS-485.

4.4.8. Projeto SCADDS

O *Scalable Coordination Architectures for Deeply Distributed Systems*(SCADDS) [ISI, 2004]é um projeto de pesquisa da USC/ISI que visa abordar arquiteturas escaláveis de coordenação em sistemas distribuídos e dinâmicos, em especial, RSSFs. Esse projeto foi motivado e elaborado a partir do artigo [Estrin et al., 1999] e envolve localização, sincronização de relógio,autoconfiguração e comunicação em RSSFs. O projeto não propõe arquitetura de hardware e nem utiliza uma em especial. Dentre as utilizadas como plataforma de teste está a baseada no Mica Motes, abordada nesteminicurso.Quanto aos protocolos de comunicação utilizados, destacam-se o S-MAC [Ye et al., 2002] (ver seção 4.3.2.1), na camada de enlace, e o Direct Diffusion [Intanagonwiwat et al., 2002] (ver seção 4.3.3.1, na camada de rede. O S-MAC adota um esquema TDMA para comunicação entre vizinhos, o que representa uma vantagem em RSSFs por permitir economia de energia nos instantes onde não há comunicação. Esse protocolo tenta atender alguns requisitos de dinamicidade da rede, como inclusão de nós e tolerância a falhas, porém parece não atender bem redes com nós móveis. O Direct Diffusion, tradicional

algoritmo de disseminação de dados em RSSFs, propõe um esquema de roteamento centrado em dados, onde não há semântica de endereçamento. Ele também tenta atender redes dinâmicas através de um esquema de negociação com disseminação de interesses e reforços de caminhos, permitindo à rede convergir perante qualquer alteração topológica. Sua grande desvantagem é o alto custo de comunicação, devido à necessidade de se disseminar interesses periodicamente em toda a rede, e disseminar dados até que um caminho seja reforçado. Em simulações realizadas pelo grupo de pesquisa do Projeto SensorNet da UFMG [SensorNet, 2003], foi demonstrado que o algoritmo sofre muito com perdas de pacotes quando a quantidade de nós enviando dados começa a aumentar e a taxa de envio de dados é alta. Com uma pilha montada desta forma, se tem a vantagem de trabalhar com uma plataforma projetada especialmente para RSSFs, porém, observa-se a limitação a aplicações de redes estáticas e com disseminação de dados sob requisição (interesses). Logo, redes de tráfego contínuo e, principalmente, orientadas a eventos, devem empregar outros algoritmos da camada de rede.

4.4.9. Projeto MANTIS

O Multimodal Networks of In-situ Sensors (MANTIS) [Colorado UC, 2004] é um projeto do Departamento de Computação da Universidade do Colorado focado na pesquisa em RSSFs que envolve hardware e sistema operacional, como o desenvolvimento do nó sensornymph e do MANTIS OS [Abrach et al., 2003], protocolos de comunicação, ferramentas e aplicações. O protocolo 802.11 [IEEE 802.11, 2003] é utilizado na camada de enlace, o que acarreta em alto consumo de energia. Para tentar reduzir esse problema, foi introduzido um esquema de controle de potência e ativação seletiva do rádio, onde se regula a potência de transmissão e seletivamente se faz o rádio de um nó dormir para maior economia de energia [Sheth and Han, 2003].Para o roteamento de dados, foi desenvolvido o algoritmo VLM2 [Sheth et al., 2003], que pretende ser um algoritmo leve, ou seja, que consume pouco os recursos da rede, e que introduz a capacidade de comunicação em *multicast*. O algoritmo de roteamento é baseado na construção de uma árvore a partir de um nó raiz (nó sorvedouro), e portanto,não há comunicação ponto-a-ponto entre nós. Para acomodar mudanças topológicas e nova criação de grupos de *multicast*, a árvore deve ser periodicamente refeita. Soluções baseadas em árvores de disseminação de dados são simples e bastante adotadas em RSSFs, tendo sido avaliadas em projetos de pesquisa como o SensorNet [SensorNet, 2003]. Esta soluções apresentam desempenho melhor que outras soluções na simples tarefa de entregar dados ao nó raiz. Para a solução intermediária adotada pela plataforma MANTIS é esperado um desempenho inferior a outras soluções baseadas em TDMA, mas que só um trabalho de avaliação poderá confirmar esta suposição. De qualquer maneira, a solução intermediária não é adequada para aplicações orientadas a eventos ou de tráfego sob demanda, pois tanto a solução de MAC quanto de roteamento mantém atividade em períodos ociosos da rede.

4.4.10. Projeto SensoNet

O SensoNet [GATECH, 2004] é um projeto do Instituto de Tecnologia da Georgia com o propósito específico de desenvolver protocolos de comunicação para as RSSFs e suas características particulares. Este projeto, que usa o Mica Motes como plataforma de teste, não propõe

uma solução de pilha de protocolos específica, mas apresenta vários protocolos individualmente que poderiam compô-la, como o ESRT [Sankarasubramaniam et al., 2003], para camada de transporte, o SER [Su and Akyildiz, 2002] e o QSR [Su and Akyildiz, 2003], para roteamento e o CMAC [Vuran and Akyildiz, 2003] na camada MAC. O principal requisito visado por esses protocolos é eficiência energética e confiabilidade na entrega de dados, mas não é apresentado como as pilhas de protocolos são montadas e qual o desempenho do conjunto.

4.4.11. Projeto BEAN

Os pesquisadores Projeto SensorNet do DCC/UFMG estão desenvolvendo um nó sensor usando componentes de prateleira. No mesmo projeto está em desenvolvimento a plataforma computacional chamado de BEAN(*Brazilian Energy-Efficient Architectural Node*) [Vieira, 2004b] que servirá como protótipo de um nó sensor. O microcontrolador utilizado é da família MSP430, que é *ultra-low power*, além de ser 16bits 8 MIPS e possuir vários modo de operações e é equipado com um conjunto completo de conversor analógico-digital, facilitando a integração dos dispositivos sensores. O rádio utilizado será o CC1000 (o mesmo do Mica2 Mote). Uma memória serial flash externa (STM25P40) que serve como memória secundária também será utilizada. Outro componente utilizado é o ds2417 que servirá como um relógio de tempo real além de prover um número identificador único de 48-bits. O sistema operacional deste projeto também está sendo desenvolvido pelos pesquisadores da UFMG e foi batizado de YATOS (*Yet Another Tiny Operating System*) [Vieira, 2004a]. Ele é dedicado ao nó sensor BEAN e dirigido a eventos. Uma das vantagens em relação ao TinyOS (ver seção 4.5) é que o YATOS possui prioridade entre tarefas.

4.4.12. MillennialNet

A Millennial [Millennial Net, 2004] possui uma solução de RSSFs composta de nós com funções especiais. São eles: Endpoints, para realizar o sensoriamento;Routers, para estender a área monitorada através de *multi-hop*; e *Gateway*, para conexão da RSSF com redes externas. A idéia da rede da Millennial é a criação de uma rede auto-organizável, que automaticamente cria e mantém uma topologia de rede mesmo com a ocorrência de alterações topológicas, com tempo de vida longo(anos), exigindo operação com baixíssimo consumo de energia. A solução da Millennial é fechada, não permitindo uma análise mais profunda. Porém, algumas características nos dão algumas dicas. Entre as possibilidades de rádio está o IEEE 802.15.4 [IEEE, 2004], para redes pessoais sem fio, prometendo baixo consumo, porém, baixa largura de banda. Já no roteamento, se tem um protocolo com patente pendente com a idéia de usar a arquitetura com nós roteadores para dar maior confiabilidade à entrega de pacotes formando uma infra-estrutura em malha. O protocolo opera com baixas taxas de coleta de dados e, ao que tudo indica, simplesmente tem a função de entregar pacotes ao *gateway* pelo menor caminho (menor número de hops).Pelas poucas informações obtidas, a arquitetura fechada da Millennial se aplica somente a casos de simples coleta de dados com tráfego contínuo, não sendo possível adequá-la a soluções de processamento colaborativo distribuído, porém, já disponibiliza uma rede com longo tempo de vida.

4.4.13. Projeto PicoRadio

O PicoRadio [Pico, 2003] está sendo projetado na Universidade de Berkeley, é um tipo de nó micro sensor conhecido como picoSensor. Este tipo de dispositivo é projetado com o objetivo de que a dissipação de energia do sensor, tanto em processamento quanto em comunicação, seja extremamente baixa. Portanto, os limites aceitáveis em relação à energia são de 10pJ por bit corretamente transmitido ou processado, e em relação à potência, o máximo é de 1 mW. Para que tais objetivos possam ser alcançados, as seguintes estratégias são utilizadas: (1) *Energy sacavenging* (técnica cujo objetivo é conseguir que o nó sensor retire o máximo de energia possível do ambiente onde se encontra. Por exemplo, energia solar ou energia de vibrações; (2) baixo consumo de energia na arquitetura do PicoSensor e seus circuitos, ou seja, projeto e desenvolvimento de componentes de baixo consumo de energia; (3) projeto de sistema operacional dirigido a eventos (resultados de testes mostraram que este tipo de sistema pode ser mais econômico do que sistemas operacionais de propósito geral, no que diz respeito a energia). A largura de banda é de 5 GHz e a camada de enlace (MAC) utiliza TDMA.

4.5. Sistema Operacional TinyOS

Esta seção apresenta um descrição do sistema operacional TinyOS, incluindo suas funcionalidades e arquitetura. O TinyOS foi desenvolvido pelo Departamento de Engenharia Elétrica e Ciência da Computação (EECS) da Universidade da Califórnia - Berkeley para trabalhar com o microprocessador ATMEL AT[4]. Trata-se de um sistema operacional muito simples e compacto, baseado em eventos, desenvolvido para atender alguns dos requisitos das RSSFs, quais sejam operações intensivas de concorrência com requisitos mínimos de hardware e para a economia de energia. A linguagem de programação utilizada no TinyOs é uma linguagem C estilizada chamada NesC (pronunciado “NES-see”) .

4.5.1. História do TinyOS

O TinyOS foi inicialmente desenvolvido por Jason Hill, em seu projeto de Mestrado - UC Berkeley - 2000 [Hill, 2000]. Ele foi orientado pelo Professor David Culler em um projeto que involve a Universidade de Berkeley e a Intel [Berkeley, 2003]. O TinyOS faz parte do Projeto Berkeley WEBS (*Wireless Embedded System*). Outros trabalhos de RSSFs que estão sendo desenvolvidos pelo mesmo grupo são: TOSSIM (um simulador para o TinyOS), Maté (uma máquina virtual para o TinyOS), TinyDB (um sistema de processamento de consultas desenvolvido para extrair informação de uma RSSF utilizando TinyOS), NesC (um compilador personalizado específico para o TinyOS), Calamari (um sistema de localização para redes de sensores), *Great Duck Island* (uma aplicação que emprega nós sensores chamados de motes para monitorar habitats e vida selvagem) e TinySEC (uma camada de enlace com criptografia voltada para pequenos dispositivos). Atualmente, as modificações no código e nos componentes do TinyOS são feitas pelo grupo Intel-Berkeley Research Lab.

[4]©Atmel Corporation

4.5.2. O que é o TinyOS?

TinyOS é composto por um sistema operacional simples, um ambiente de desenvolvimento com código aberto, um modelo e uma linguagem de programação. O sistema operacional utiliza a eventos e um conjunto de serviços, como descrito a seguir.

O TinyOS é um sistema operacional simples. Ele possui um escalonador de tarefas, que é uma fila (FIFO – "*First In First Out*"), utilizando uma estrutura de dados de tamanho limitado. O escalonador, desenvolvido para concorrência intensiva, é não preemptivo e não possui mecanismos sofisticados como fila de prioridades. Os nós sensores estarão monitorando eventos do mundo real, e estes são inerentemente concorrentes (mais de um evento pode ocorrer em um mesmo intervalo de tempo). Como vários eventos podem ocorrer, as tarefas que vão atender a estes eventos têm que executar eficientemente e o sistema operacional tem que ser projetado para que todos os eventos possam ser atendidos a tempo. Além disso, os recursos computacionais em um nó sensor são limitados. Dessa forma, o TinyOS também foi construído para trabalhar com recursos limitados e facilitar o desenvolvimento de componentes de software voltados para eficiência e modularidade.

O TinyOS possui um conjunto de ferramentas que permitem desenvolver código para este sistema. O código fonte é aberto, e pode ser baixado no site: `http://today.cs.berkeley.edu/tos/`

O TinyOS também é um modelo e uma linguagem de programação. Programas são construídos a partir de um conjunto de componentes. A especificação do comportamento dos componentes é feito através de um conjunto de interfaces. Componentes são estaticamente conectados um ao outro via interfaces. Isto aumenta a eficiência em tempo de execução porque as conexões entre componentes são checadas em tempo de compilação.

O TinyOS possui um modelo de eventos que permite ter concorrência utilizando pouco espaço de memória. Um modelo baseado em troca de contexto e pilha iria requerer que o espaço da pilha fosse reservado para cada troca de contexto, além de requerer processamento em cada troca. Além disso, a energia é um recurso precioso. Conforme descreve Suet-Fei et. al. [Li et al., 2001], o modelo baseado em eventos alcança um ganho de 12 vezes comparado com o modelo com troca de contexto. Os ciclos não utilizados da CPU são utilizados no estado "*sleep*" , ao invés de procurar ativamente um evento.

Finalmente, o TinyOS também é um conjunto de serviços. Interfaces e componentes que implementam os principais serviços de uma aplicação de RSSF estão disponíveis junto com o TinyOS. Estes serviços são:

- Rádio, MAC, Mensagens, Roteamento: componentes que implementam a pilha de protocolos do TinyOS.
- Interface de Sensores: conjunto de interfaces para os mais variados tipos de sensores que podem ser utilizados em um nó sensor com TinyOS.
- Gerência de Energia: energia é o recurso mais crítico de uma RSSFs.
- Depuração: provê componentes e ferramentas que permitem a depuração de código.
- Temporizadores: provêem componentes de acesso aos relógios do nó sensor.

4.5.3. Objetivos do TinyOS

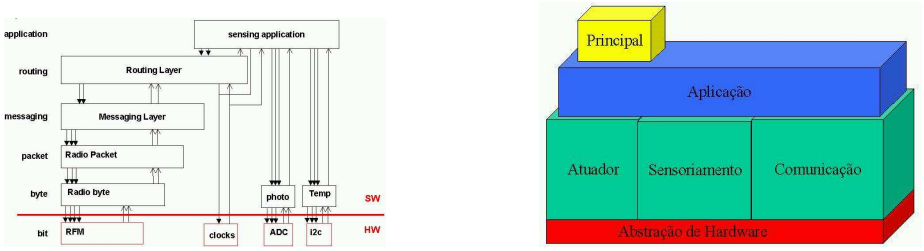
Os objetivos do TinyOS são:

- Atender sistemas embutidos em redes: o sistema deveria "dormir", mas permanecer vigilante a estímulos. Quando um ou mais eventos ocorressem, estes deveriam ser atendidos.
- Mica hardware: o TinyOS é voltado especificamente para a arquitetura Mica Motes. Tarefas e componentes de energia, sensoriamento, computação e comunicação dessa plataforma são implementados no TinyOS.
- Acompanhar os avanços tecnológicos: prevendo um desenvolvimento da tecnologia de MEMS, o TinyOS deve acompanhar os avanços tecnológicos, para que não fique ultrapassado rapidamente. A tendência dos circuitos integrados é de continuar a reduzir as dimensões de tamanho, ficando mais barato, e incluindo tecnologias de pouco consumo de energia (low power).

RSSFs não podiam utilizar sistemas operacionais de tempo real (RTOS) existentes. A arquitetura desses sistemas como VxWorks [VxWorks 5.4, 2003], QNX [Hildebrand, 2003], WinCE [Microsoft Windows CE, 2003], PalmOS [PalmOS, 2003], μLinux [microLinux, 2003] eram similares a de Desktops. Sistemas Operacionais voltados para PDA's, telefones celulares, sistemas embutidos baseados na arquitetura PC eram mais de um ordem de magnitude pesados e lentos, sem contar o consumo proibitivo de energia.

4.5.4. Eventos, Comandos e Tarefas

TinyOS é um modelo e uma linguagem de programação. Ele representa um novo paradigma de programação, incluindo conceitos de eventos, comandos e tarefas. Uma configuração completa do sistema consiste em um minúsculo programa composto de uma aplicação e dos componentes do TinyOS. Uma aplicação é, na verdade, um conjunto de componentes agrupados, conforme mostrado na figura 4.15(a). Estes também podem ser agrupados em camadas, como ilustra a figura 4.15(b).



(a) (b)

Figura 4.15: Uma aplicação pode ser vista como um grupo de componentes.

Um componente possui quatro partes relacionadas: um conjunto de comandos, um conjunto de tratadores de eventos, um "*frame*" encapsulado de tamanho fixo e um pacote de tarefas

simples. As tarefas, os comandos, e os tratadores de eventos executam no contexto do "*frame*" e operam sobre seu estado. Para facilitar a modularidade, cada componente também declara os seus comandos e os eventos que sinaliza. Estas declarações são usadas para compor os componentes. O processo de composição cria camadas de componentes, onde os componentes de alto nível emitem comandos aos componentes de baixo nível e os componentes de baixo nível sinalizam eventos aos componentes de alto nível. A parte física de hardware representa o nível mais baixo dos componentes.

Os *frames* possuem tamanho fixo e são alocados estaticamente, permitindo conhecer os requerimentos da memória de um componente em tempo de compilação, o que evita o custo adicional associado com alocação dinâmica. A alocação estática de memória também diminui o tempo de execução, uma vez que as posições das variáveis podem ser definidas estaticamente no programa. Os comandos são pedidos assíncronos feitos aos componentes do nível inferior. Um comando pode colocar uma tarefa para executar futuramente. Ele pode também invocar alguns comandos de níveis mais baixos, mas não deve esperar executar ações longas ou de latência indeterminadas. Um comando deve fornecer uma resposta ao seu chamador, indicando se foi bem sucedido ou não. Já os tratadores de eventos são invocados para tratar dos eventos do hardware, diretamente ou indiretamente. Os componentes de baixo nível têm os tratadores conectados diretamente às interrupções do hardware, que podem ser interrupções externas (eventos do temporizador) ou eventos contadores. Um tratador de eventos pode iniciar tarefas, sinalizar eventos de alto nível ou chamar comandos do nível inferior. Um evento de hardware provoca uma seqüencia de processamento que sobe de nível através dos eventos e pode descer através dos comandos (veja a figura 4.16). Com a finalidade de evitar ciclos em cadeia de comandos/eventos, os comandos não podem sinalizar eventos.

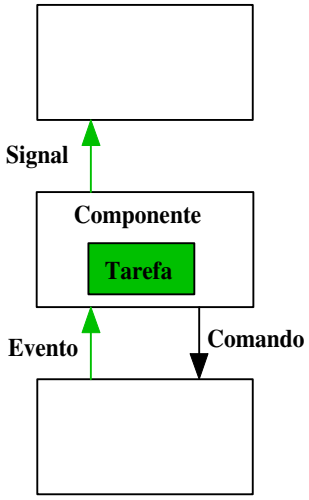


Figura 4.16: Tratador.

As tarefas executam trabalhos básicos, sendo atômicas com respeito a outras tarefas e executadas completamente. As tarefas podem chamar comandos de nível inferior, sinalizar eventos de um nível superior e programar outras tarefas dentro de um componente. A execução completa das tarefas torna possível alocar uma única pilha, que é atribuída à tarefa que é processada no momento. Isto é essencial em sistemas com restrição de memória. As tarefas permitem a simulação de concorrência dentro de cada componente, desde que executadas de forma assíncrona em relação aos eventos. Entretanto, as tarefas nunca devem obstruir ou prolongar a espera, pois bloquearão outros componentes.

O escalonador de tarefa é uma fila FIFO, utilizando uma estrutura de dados de tamanho limitado. O processador entra no modo "*sleep*" toda vez que a fila de tarefas estiver vazia, mas os periféricos continuam em operação, de modo que alguns deles podem reativar o sistema. Este comportamento permite o uso eficiente da energia. Uma vez que a fila está vazia, uma outra tarefa pode ser programada somente em conseqüência de um evento, assim não há nenhuma ne-

cessidade para o escalonador “despertar” até que um evento do hardware provoque uma atividade. A aplicação também é responsável pela gerência de energia, que é o recurso mais crítico em RSSF.

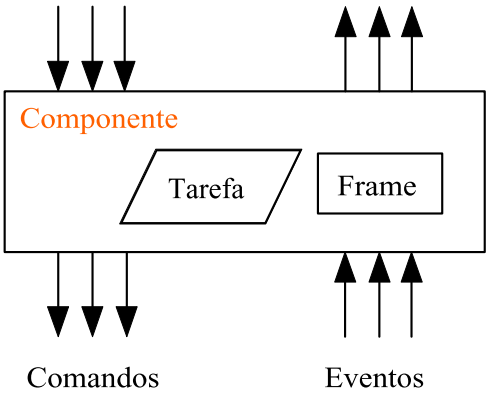


Figura 4.17: Componente.

A figura 4.17 retrata o conceito de componente. O componente possui uma interface com métodos definidos (representado na figura por setas), que o permite agrupar com outros componentes tanto no sentido para cima, como para baixo. Um componente típico inclui um *frame*, tratadores de evento, comandos e tarefas para um componente de manipulação de mensagem. Como a maioria dos componentes, este exporta comandos de inicialização e gerenciamento de energia; possui um comando para iniciar uma transmissão de mensagem, e sinaliza eventos na conclusão da transmissão ou na chegada de uma mensagem. Os componentes de transmissão de mensagem editam comandos no pacote no nível de componente e definem dois tipos de eventos. O primeiro indica que a mensagem foi transmitida e o segundo sinaliza que a mensagem foi recebida. A conexão entre os componentes é bastante simples, desde que os componentes descrevam os recursos que fornecem e os recursos que requerem. Assim, o programador combina simplesmente as assinaturas dos eventos e dos comandos requeridos por um componente com as assinaturas dos eventos e dos comandos fornecidos por um outro componente. A comunicação através dos componentes examina um formulário de chamada de função, que possui os dados necessários para a comunicação e fornece a verificação do tipo em tempo de compilação.

4.5.5. O Projeto do Kernel TinyOS

O projeto do kernel do TinyOS é baseado numa estrutura de dois níveis de escalonamento:

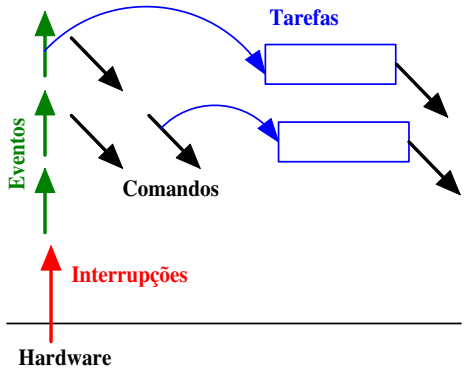


Figura 4.18: Fluxo de execução.

1. Eventos utilizam uma pequena quantidade de processamento (interrupções de relógio, interrupções ADC) e podem interromper tarefas que estejam em execução.
2. Tarefas utilizam uma quantidade maior de processamento e não são críticas quanto ao tempo como eventos. Tarefas sempre executam até completarem. Esta propriedade é muito importante, pois permite que o TinyOS utilize apenas uma pilha de execução.

A figura 4.18 ilustra o fluxo de execução de tarefas (retângulos), eventos (setas para cima) e comandos (setas na diagonal). Interrupções são mapeadas como

eventos. Um evento pode acionar comandos ou acionar a execução de uma tarefa. Uma tarefa é formada por um conjunto de comandos e não pode bloquear outra tarefa, mas um evento pode interromper a execução de uma tarefa. Na figura 4.18, uma interrupção gera um evento. Este gera um evento para o nível superior, que por sua vez executa um comando e gera um evento para o nível superior. O tratador de eventos executa dois comandos, e cada um inicia uma tarefa, que executam um conjunto de comandos. Eventos gerados por interrupção interrompem tarefas, enquanto tarefas não interrompem tarefas. Tarefas e tratadores de eventos executam um conjunto de comandos.

A tabela 4.8 resume as estruturas usadas no escalonador do kernel.

Item	**Eventos**	**Tarefas**
Processamento	Pequena quantidade de processamento	Não são críticas em relação ao tempo
Exemplos	Relógios e Interrupções; ADC (conversão analógico digital)	Calcular a média de um vetor
Propriedades	Podem interromper tarefas executando	Executam até finalizar

Tabela 4.8: Eventos versus Tarefas

4.6. Ambiente de Programação

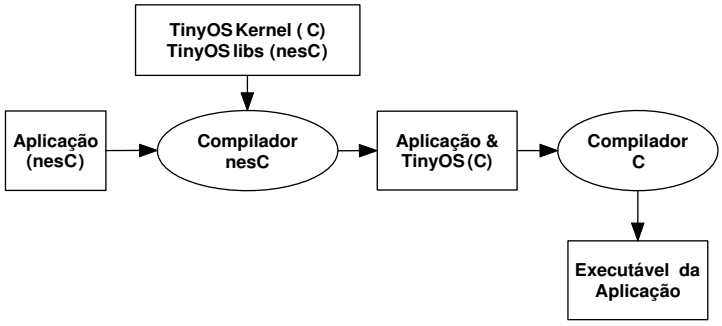


Figura 4.19: Etapas envolvidas no desenvolvimento de uma aplicação no TinyOS.

Esta seção apresenta uma breve descrição das ferramentas de software para desenvolvimento de aplicações para RSSFs, incluindo uma descrição da linguagem de programação NesC e ferramentas de simulação TOSSIM e TinyViz.

As etapas que envolvem o desenvolvimento de uma aplicação no TinyOS são mostradas na figura 4.19. Uma aplicação é descrita na linguagem NesC. Em seguida, o código fonte desta aplicação, junto com o kernel do TinyOS e as bibliotecas do TinyOS são compilados por um compilador NesC, resultando no código de uma aplicação com TinyOS em C. Este código é compilado por um compilador C, gerando um executável de uma aplicação. Os tipos de arquivos em cada uma das etapas do desenvolvimento de aplicação possuem extensões diferentes, que servem para identificar o formato do arquivo. A figura 4.20 apresenta o fluxo de compilação. Arquivos escritos em NesC possuem extensão `.nc`. Quando compilados através do compilador da linguagem NesC, o `ncc`, geram arquivos em C que possuem extensão `.c`.

Um compilador C (no caso o `avr-gcc`) gera código de máquina (extensão `.s`) que serve de entrada para um montador (no caso o `avr-as`). O montador irá gerar o código objeto (extensão `.o`). Por fim, o compilador `avr-gcc` é utilizado para gerar o código executável (extensão `.exe`). Este pode ser carregado no hardware do Mica Motes no formato S-Records (arquivos com extensão `.srec`). A conversão do executável para o formato S-Records é feita pelo `avr-objcopy`.

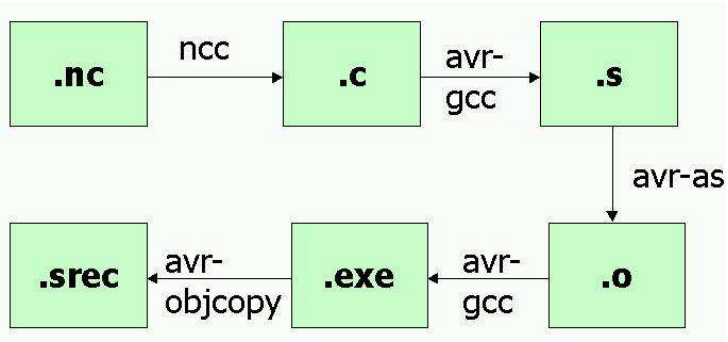


Figura 4.20: Etapas de compilação.

Esclarecido o processo de geração de código, é preciso compreender os conceitos da linguagem para poder desenvolver aplicações. Dessa forma, NesC é brevemente explicada a seguir.

4.6.1. NesC

A linguagem NesC [Culler et al., 2003] é uma extensão da linguagem de programação C projetada para incluir os conceitos estruturais e modelos de execução do TinyOS. Conforme mencionado anteriormente, o TinyOS é um sistema operacional dirigido a eventos voltado para RSSF que possui recursos limitados (por exemplo, 8 Kbytes de memória de programa, 512 bytes of RAM). O manual da linguagem [nesC, 2003], assim como o código fonte, são abertos e podem ser encontrados no site: `http://nescc.sourceforge.net/`.

4.6.2. Conceitos Básicos do NesC

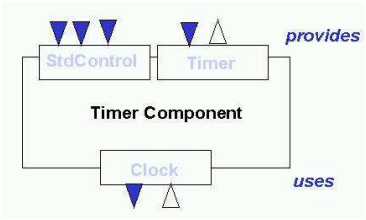


Figura 4.21: Um componente e sua interface.

O NesC permite separar construção de composição. Aplicativos escritos em NesC são compostos por componentes, que podem ser construídos e combinados para formar uma aplicação, aumentando a modularidade e reusabilidade de código. Em NesC, o comportamento de um componente é especificado em termos de um conjunto de interfaces. Interfaces são bi-direcionais, e informam o que um componente usa e o que ele provê. A figura 4.21 ilustra um componente (Timer) e sua interface.

Componentes são estaticamente ligados um ao outro via interfaces. O fluxo de informação pode ocorrer com camadas inferiores (via comandos) ou com camadas superiores (via eventos), conforme mostra a figura 4.16. Conceitos do TinyOS, como eventos, tarefas e comandos, estão embutidos no NesC. Relembrando os pontos importantes destes conceitos:

1. Tarefas
    - Realizam computação (conjunto de comandos)
    - Não são críticas com relação a tempo
2. Eventos
    - Críticas em relação a tempo
    - Sinalizam interrupções externas
    - Geram um "Signal"
    - Receptor recebe/aceita um "Event"
3. Comandos
    - Funções de procedimentos para outros componentes
    - Não podem sinalizar eventos

O TinyOS provê outras ferramentas para facilitar o desenvolvimento de aplicações, como simuladores e depuradores. A seguir são descritos os simuladores do TinyOS.

4.6.3. O Simulador TOSSIM

O TOSSIM (The TinyOS simulator) [Levis and Lee, 2003] é um simulador discreto de eventos para RSSFs que usam o TinyOS. Ao invés de compilar a aplicação do TinyOS para o mote, usuários podem compila-lá para o ambiente do TOSSIM, que executa em um PC. O TOSSIM permite que usuários depurem, testem, e analisem algoritmos em ambientes controlados. Como o TOSSIM executa em um PC, usuários podem examinar seus códigos TinyOS utilizando depuradores e outras ferramentas de desenvolvimento utilizadas em programas para PC.

O objetivo primário do TOSSIM é prover uma simulação com alta fidelidade das aplicações para o TinyOS. Por esta razão, TOSSIM prefere focar na simulação do TinyOS e na execução deste a simular o mundo real. Enquanto o TOSSIM pode ser usado para compreender as causas de comportamentos observados no mundo real, ele não captura todos estes, e por isso não deve ser usado para avaliações absolutas. O código que é executado no TOSSIM é compilado diretamente do código TinyOS. Este código pode ser executado nativamente em um desktop ou laptop. TOSSIM permite simular milhares de nós sensores simultaneamente. Na simulação, cada nó sensor executa o mesmo programa TinyOS.

4.6.4. O Ambiente TinyViz

TinyViz é um programa em Java com interface gráfica que permite visualizar e controlar a simulação enquanto ela é executada, inspecionando mensagens de depuração, rádio e pacotes UART (*Universal Asynchronous Receiver Transmitter*).

TinyViz provê vários mecanismos de interação com a rede. O TinyViz provê suporte à monitoração de tráfego de pacotes e injeção de pacotes na rede de forma dinâmica. A figura 4.22 mostra a visualização (interface gráfica) do TinyViz.

4.7. Desenvolvendo uma Aplicação

Nesta seção é desenvolvida uma aplicação simples que utiliza os conceitos, componentes e ferramentas descritos nas seções anteriores. O NesC provê sintaxe para o modelo do TinyOS,

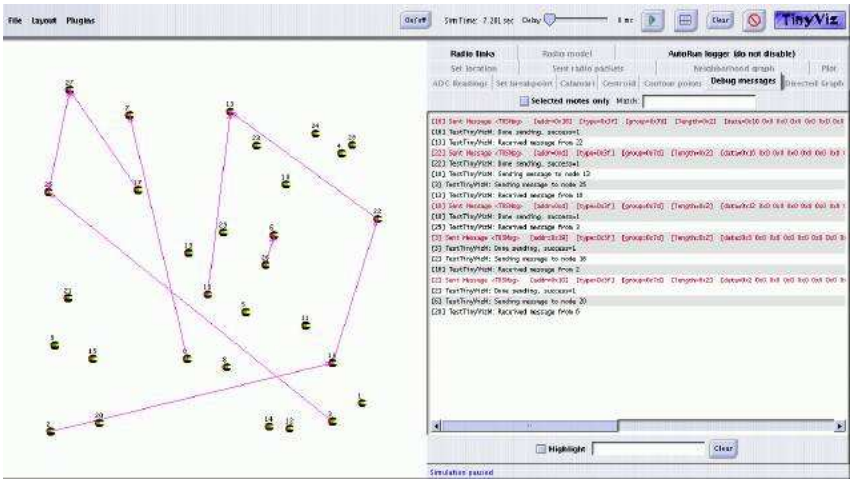


Figura 4.22: Um exemplo de janela da interface gráfica do Tinyviz.

criando comandos, eventos e tarefas. O NesC utiliza o conceito de Interfaces para aumentar o reuso. Interfaces especificam a funcionalidade de um componente para o mundo exterior. Elas identificam quais comandos podem ser chamados e quais eventos precisam ser tratados. Convencionalmente, arquivos de interface são nomeados "*.nc". Os componentes de software são formados por módulos e configurações. Os arquivos "*M.nc" possuem a e a definição de interface dos módulos. Os arquivos de configuração são nomeados "*C.nc", e conectam componentes (módulos e configurações) em unidades. Opcionalmente, os arquivos de configuração especificam as interfaces que usam e provem. O mais importante é que eles não possuem código C. Quando representarem a aplicação no nível mais alto, podem perder a letra C do nome do arquivo.

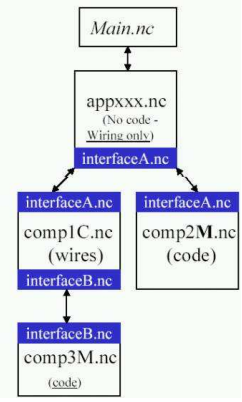


Figura 4.23: Divisão.

Um exemplo simples de programa que faz um LED do Mote piscar e que é executado com o TinyOS será explicada nesta seção. Trata-se da aplicação chamada "Blink" que pode ser encontrada no caminho "apps/Blink" da árvore do TinyOS. Esta aplicação causa o LED vermelho do mote acender e apagar em uma freqüência de 1 Hz.

A aplicação Blink é composta de dois componentes: um módulo denominado "BlinkM.nc", e uma configuração chamada "Blink.nc". É importante ressaltar que todas as aplicações requerem um arquivo de configuração no nível mais alto, que tipicamente possui o nome da própria aplicação. Neste caso, o arquivo "Blink.nc" é a configuração para a aplicação Blink, que é utilizada pelo compilador NesC para gerar o arquivo executável. Este arquivo também conecta o módulo BlinkM.nc aos outros componentes que aplicação Blink utiliza. "BlinkM.nc" provê a implementação da aplicação. A figura 4.23 ilustra os componentes de uma aplicação e a conexão entre interfaces.

A distinção entre os módulos e configurações permite a um desenvolvedor de sistema o

desenvolvimento de aplicações através da conexão de módulos, o que permite uma programação ágil. Por exemplo, um desenvolvedor pode prover uma configuração que simplesmente conecta um módulo a outros módulos, nenhum dos quais ele desenvolveu. Da mesma forma, outro desenvolvedor pode prover um novo conjunto de bibliotecas de módulos que podem ser usados em diversas aplicações.

Em alguns casos (como é no caso do Blink e BlinkM), uma configuração e um módulo podem ser unidos. Quando este é o caso, a convenção usada na árvore fonte do TinyOS é que "Foo.nc" representa uma configuração e "FooM.nc" representa o módulo correspondente. Um resumo das convenções de nomes utilizados no código do TinyOS se encontra no site `http://today.cs.berkeley.edu/tos/tinyos-1.x/doc/tutorial/naming.html`.

4.7.1. O Arquivo de Configuração Blink.nc

O compilador da linguagem NesC, o `ncc`, compila uma aplicação escrita em NesC quando o arquivo dado for de uma configuração de nível mais alto. Tipicamente, aplicações do TinyOS apresentam um "Makefile" padrão, que permite escolher a plataforma alvo e invocar o `ncc` com as opções apropriadas. A seguir é mostrado o arquivo de configuração da aplicação Blink.

```
configuration Blink {
}
implementation {
  components Main, BlinkM, SingleTimer, LedsC;

  Main.StdControl -> BlinkM.StdControl;
  Main.StdControl -> SingleTimer.StdControl;
  BlinkM.Timer -> SingleTimer.Timer;
  BlinkM.Leds -> LedsC;
}
```

A primeira observação é a palavra chave "*configuration*", que indica que este arquivo é de configuração. As duas primeiras identificam que esta é a configuração da aplicação chamada Blink. Dentro do par de chaves vazia é possível especificar cláusulas "*uses*" e "*provides*", assim como em um módulo. Esta é uma observação importante: uma configuração pode usar e prover interfaces.

A configuração é implementada dentro do par de chaves seguido da palavra chave implementation. A linha *components* especifica o conjunto de componentes que a configuração referencia, no nosso caso *Main*, *BlinkM*, *SingleTimer* e *LedsC*. O restante da implementação consiste em conectar as interfaces usadas pelo componente com a interface provida pelos outros componentes.

Em uma aplicação TinyOS, o componente que é executado primeiro é o *Main*. Mais precisamente, o comando `Main.StdControl.init()` é o primeiro a ser executado no TinyOS, seguido do `Main.StdControl.start()`. Uma aplicação TinyOS deve possuir um componente Main na sua configuração. *StdControl* é a interface comum usada para inicializar os componentes do TinyOS. O próximo passo é investigar o arquivo `tos/interfaces/StdControl.nc`:

```
interface StdControl {
  command result_t init();
  command result_t start();
  command result_t stop();
}
```

StdControl define três comandos, `init()`, `start()`, e `stop()`. `init()` é chamado quando um componente é inicializado pela primeira vez, e `start()` é chamado quando o componente é posto a executar pela primeira vez. `stop()` é chamado quando um componente é parado, por exemplo, com o objetivo de desligar o dispositivo que está sendo controlado para economizar energia. `init()` pode ser chamado múltiplas vezes, mas nunca depois de chamar `start()` ou `stop()`. Especificamente, a expressão regular que representa o padrão válido de chamadas do StdControl é `init*(start|stop)*`. As duas linhas seguintes no arquivo de configuração Blink conectam a interface *StdControl* no *Main* com a interface *StdControl* em ambos *BlinkM* e *SingleTimer*.

```
  Main.StdControl -> SingleTimer.StdControl;
  Main.StdControl -> BlinkM.StdControl;
```

`SingleTimer.StdControl.init()` e `BlinkM.StdControl.init()` serão chamados pelo `Main.StdControl.init()`. A mesma regra se aplica aos comandos `start()` e `stop()`.

A respeito das interfaces usadas (palavra chave *uses*), é importante ressaltar que as funções de inicialização de subcomponentes devem ser explicitamente chamadas pelo componente usado. Por exemplo, o módulo BlinkM usa a interface Leds, portanto `Leds.init()` deve ser chamada explicitamente em `BlinkM.init()`.

NesC usa setas para determinar relacionamentos entre interfaces. Pense na seta a direita ($->$) como "ligando a". O lado esquerdo da seta liga uma interface a uma implementação no lado direito, ou seja, o componente que usa (palavra chave *uses*) uma interface está na esquerda, e o componente que fornece (palavra chave *provides*) a interface está na direita.

A linha abaixo é usada para conectar a interface Timer usada pelo BlinkM a interface Timer fornecida pelo `SingleTimer`.

```
  BlinkM.Timer -> SingleTimer.Timer;
```

Conexões também são implícitas. Por exemplo,

```
  BlinkM.Leds -> LedsC;
```

é equivalente a

```
  BlinkM.Leds -> LedsC.Leds;
```

4.7.2. O Módulo BlinkM.nc

```
module BlinkM {
  provides {
    interface StdControl;
```

```
  }
  uses {
    interface Timer;
    interface Leds;
  }
}//Continua abaixo
```

A primeira parte do código indica que este é um módulo chamado BlinkM e declara as interfaces fornecidas e utilizadas. O módulo BlinkM fornece a interface `StdControl`, o que significa que BlinkM implementa a interfaces de `StdControl`. Como explicado anteriormente, isto é necessário para iniciar o componente Blink e executá-lo. O módulo BlinkM também utiliza duas interfaces: Leds (diôdos emissores de luz) e Timer (temporizador). Isto significa que BlinkM pode chamar qualquer comando declarado nestas interfaces, e deve também implementar quaisquer eventos declarados por estas interfaces.

A interface Leds define alguns comandos, como `redOn()` e `redOff()`, que acendem ou desligam os diferentes LEDs (vermelho, verde, ou amarelo) do mote. BlinkM pode invocar qualquer um desses comandos porque ele usa a interface Leds.

Timer é um temporizador. O comando `start()` é usado para especificar o tipo de temporizador e o seu intervalo, especificados em milisegundos. O tipo TIMER_REPEAT indica que o temporizador continua enquanto não for parado pelo comando `stop()`. Quando o timer expira, ele gera um evento que é recebido pela aplicação. A interface Timer fornece o evento:

```
  event result_t fired();
```

Eventos podem ser vistos como uma função de retorno que a implementação de uma interface irá invocar. Um módulo que usa uma interface deve implementar os eventos que esta interface usa.

BlinkM.nc, continuação

```
implementation {
  command result_t StdControl.init() {
    call Leds.init();
    return SUCCESS;
  }

  command result_t StdControl.start() {
    return call Timer.start(TIMER_REPEAT, 1000) ;
  }

  command result_t StdControl.stop() {
    return call Timer.stop();
  }

  event result_t Timer.fired()  {
    call Leds.redToggle();
```

```
    return SUCCESS;
  }
}
```

O módulo BlinkM implementa os comandos `StdControl.init()`, `StdControl.start()` e `StdControl.stop()`. Ele também implementa o evento `Timer.fired()`, que é necessário uma vez que BlinkM deve implementar todos os eventos das interfaces utilizadas.

O comando `init()` inicializa o sub-componente Leds. O comando `start()` invoca `Timer.start()` no intuito de criar um temporizador cíclico que expira a cada 1000 ms. O comando `stop()` termina o temporizador. Cada vez que o temporizador expira, um evento é gerado e `Timer.fired()` é acionando, modificando o estado do diodo vermelho. A linha `Leds.redToggle()` apaga o led se ele está acesso ou o acende se ele está apagado.

4.7.3. Compilando a Aplicação

Com o nó sensor conectado ao computador e utilizando um terminal aberto no diretório de instalação do TinyOS é possível compilar e carregar a aplicação no hardware.

Para compilar a aplicação Blink para o Mica Motes digite:

```
make mica
```

O makefile irá executar automaticamente os aplicativos descritos na figura 4.20. Estes também podem ser invocados na linha de comando.

O compilador NesC é acionando quando se digita:

```
ncc -o main.exe -target=mica Blink.nc
```

A aplicação Blink é compilada, e é gerado um arquivo executável `main.exe` para o Mica motes. Antes de carregar o código no mote, use

```
avr-objcopy --output-target=srec main.exe main.srec
```

para produzir o arquivo `main.srec`, que é o arquivo usado para programar o mote.

4.7.4. Gerando a Documentação

Figura 4.24: Documentação gerada para aplicação Blink.

Também é possível gerar documentação e diagramas que representam a conexão entre os componentes. O comando `make <plataform> docs` gera documentação e o diagrama de componentes da aplicação. A figura 4.24 mostra o diagrama de componentes para a aplicação do nosso exemplo.

4.8. Conclusão

A área de RSSFs tem recebido bastante atenção da comunidade de pesquisa pois propõe novos desafios e oportunidades em diferentes áreas do saber. No futuro, o sensoriamento remoto tornar-se-á parte de nossas vidas e será usado em uma variedade de aplicações.

As RSSFs são dependentes da aplicação. Assim, o projeto e desenvolvimento dos componentes de uma RSSF está diretamente ligada à aplicação que se deseja desenvolver. Existem nós sensores que, dadas as suas dimensões, taxa de transmissão e alcance, por exemplo, são ideais para uma aplicação e totalmente inadequados para outras. Em outros casos, nós que parecem adequados a um tipo de aplicação no que diz respeito ao hardware, mas apresentam limitações quanto ao software que se quer utilizar.

Este capítulo apresentou os diferentes protocolos para RSSFs propostos na literatura e alguns dos principais projetos de plataformas de nós sensores. Analisando as plataformas, observamos que muitos dos trabalhos existentes ainda não chegaram de fato a compor pilhas de protocolos. Até o momento, tem prevalecido o esquema 802.11 pela sua popularidade alcançada e roteamentos simplificados pelo esquema em árvore, embora, demonstre este ser eficiente em RSSFs para atividades simples de disseminação de dados a um sink. Assim, espera-se que novas plataformas se beneficiem de esquemas de MAC TDMA e esquemas de roteamento que permitam cooperação entre nós, como processamento distribuído de dados. Vale a ressalva de que os protocolos a serem utilizado devam se "encaixar" na formação de pilhas para atender a aplicações com requisitos específicos, principalmente no esquema de coleta de dados, tais como baseada a eventos.

Este capítulo também apresentou as principais características do sistemas operacional TinyOs e as etapas de construção e desenvolvimento de uma aplicação utilizando o ambiente do do TinyOS. Uma aplicação foi desenvolvida na linguagem NesC para mostrar este novo paradigma de programação.

Referências

Abrach, H., Bhatti, S., Carlson, J., Dai, H., Rose, J., Sheth, A., Shucker, B., Deng, J., and Han, R. (2003). Mantis: System support for multimodal networks of in-situ sensors. In *2nd ACM International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 50–59.

Badrinath, B. R., Srivastava, M., Mills, K., Scholtz, J., and Sollins, K. (2000). Special issue on smart spaces and environments. *IEEE Personal Communications*.

Beaconing, T. (2004). Tiny os: A component-based os for the networked sensor regime. `http://www.webc.cs.berkeley.edu/tos`.

Berkeley (2003). Berkeley/intel website. `http://www.intel-research.net/berkeley/index.asp`.

Bougé, L. and Frances, N. (1988). A compositional approach to superimposition. In *Proceedings of the 15th Annual ACM SIGACT/SIGPLAN Symposium On Principals of Programming Languages*, San Diego, CA, USA.

Broadwell, P., Polastre, J., and Rubin, R. (2004). Geomote: Geographic multicast for networked sensors. disponível em http://citeseer.nj.nec.com/broadwell01geomote.html.

CC 1000 (2004). Chipcom corporation. CC1000 low power FSK transceiver. http://www.chipcom.com.

CENS (2004). CENS center for embedded networked sensing. projeto medusa. http://www.cens.ucla.edu/Project-Descriptions/Sensor_Node_Plat%forms/.

Colorado UC (2004). MANTIS: Multimodal Networks of In-situ Sensors. http://mantis.cs.colorado.edu/.

Crossbow (2004). Mica2: Wireless measurement system. http://www.xbow.com/Products/New_product_overview.htm.

Culler, D., Gay, D., Levis, P., von Behren, R., Welsh, M., and Brewer, E. (2003). The nesc language: A holistic approach to networked embedded system s. In *Conference on Programming Language Design and Implementation of ACM SIGPLAN 2003*.

Dust, S. (2002). Smart Dust: Autonomous sensing and communication in a cubic millimeter. http://robotics.eecs.berkeley.edu/Epister/SmartDust.

Elson, J. and Estrin, D. (2001). Random, ephemeral transaction identifiers in dynamic sensor networks. In *Proceedings 21st International Conference on Distributed Computing Systems (ICDCS-21)*, pages 459–568, Phoenix, Arizona.

Estrin, D., Govindan, R., and Heidemann, J. (2000). Embedding the internet. *Communications of the ACM*, 43(5):39–41.

Estrin, D., Govindan, R., Heidemann, J., and Kumar, S. (1999). Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 263–270, Seattle, Washington, USA.

Figueiredo, C. M. S., Nakamura, E. F., and Loureiro, A. A. F. (2004). Protocolo Adaptativo Híbrido para Disseminação de Dados em Redes de Sensores sem Fio Auto-Organizáveis. In *Aceito para Publicação no SBRC04*.

GATECH (2004). Sensonet project: Protocols for sensor networks. http://users.ece.gatech.edu/~weilian/Sensor/index.html.

Habib, E., Câmara, D., and Loureiro, A. A. (2004). Ica: Um novo algoritmo de roteamento para redes de sensores. In *Simpósio Brasileiro de Redes de Computadores*, Gramado, RS.

Heidemann, J., Silva, F., Intanagonwiwat, C., Govindan, R., Estrin, D., and Ganesan, D. (2001). Building efficient wireless sensor networks with low-level naming. In *Proceedings of the Symposium on Operating Systems Principles*, pages 146–159, Chateau Lake Louise, Banff, Alberta, Canada. ACM.

Heinzelman, W. R., Chandrakasan, A., and Balakrishnan, H. (2000). Energy-efficient communication protocol for wireless microsensor networks. In *Proceedings of the Hawaii International Conference on System Sciences*, Maui, Hawaii, USA.

Heinzelman, W. R., Kulik, J., and Balakrishnan, H. (1999). Adaptive protocols for information dissemination in wireless sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 174–185, Seattle, WA, USA.

Hildebrand, D. (2003). An architectural overview of qnx. http://www.qnx.com/literature/whitepapers/archoverview.html.

Hill, J. (2000). A software architecture to support network sensors. Master's thesis, UC Berkeley.

IEEE (2004). Ieee 802.15 working group for wireless personal area networks. http://ieee802.org/15/index.html.

IEEE 802.11 (2003). CSMA-CA clarrier sense multiple access with collision detection. http://grouper.ieee.org/groups/802/11/.

Intanagonwiwat, C., Govindan, R., and Estrin, D. (2000). Directed diffusion: A scalable and robust communication paradigm for sensor networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 56–67, Boston, Massachusetts, USA.
Intanagonwiwat, C., Govindan, R., Estrin, D., Heidemann, J., and Silva, F. (2002). Directed diffusion for wireless sensor networking. *ACM/IEEE Transactions on Networking*, 11(1):2–16.
ISI (2004). Scadds: Scalable coordination architectures for deeply distributed systems. `http://www.isi.edu/scadds/`.
JPL (2002). JPL Sensor Webs. `http://sensorwebs.jpl.nasa.gov/`.
Kalidindi, R., Ray, L., Kannan, R., and Iyengar., S. (2003). Distributed Energy aware MAC layer for Wireless Sensor Networks. Technical report, Lousianan State University.
Karp, B. and Kung, H. T. (2000). Gpsr: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th annual international conference on Mobile computing and networking*, pages 243–254. ACM Press.
Levis, P. and Lee, N. (2003). Tossim: A simulator for tinyos networks. `http://today.cs.berkeley.edu/tos/tinyos-1.x/doc/nido.pdf`.
Li, S.-F., Sutton, R., and Rabaey, J. (2001). Low power operating system for heterogeneous wireless communication systems. In *Workshop on Compilers and Operating Systems for Low Power 2001*.
Lindsey, S., Raghavendra, C., and Sivalingam, K. M. (2002). Data gathering algorithms in sensor networks using energy metrics. *IEEE Transactions on Parallel and Distributed Systems*, 13(9):924–935.
Lindsey, S. and Raghavendra, C. S. (2002). Pegasis: Power-efficient gathering in sensor information systems. In *Proceeding of the IEEE Aerospace Conference*.
Loureiro, A. A., Ruiz, L. B., Nogueira, J. M. S., and Mini, R. A. (2002). Rede de sensores sem fio. In Porto, I. J., editor, *Simpósio Brasileiro de Computação, Jornada de Atualização de Informática*, pages 193–234.
Loureiro, A. A. F., Nogueira, J. M. S., Ruiz, L. B., de Freitas Mini, R. A., Nakamura, E. F., and Figueiredo, C. M. S. (2003). Redes de sensores sem fio. In *Simpósio Brasileiro de Redes de Computadores*, pages 179 – 226.
Macedo, D. F., Correia, L. H. A., Nogueira, J. M., and Loureiro, A. A. (2004). Proc: Um protocolo pró-ativo com coordenação de rotas em redes de sensores sem fio. In *Simpósio Brasileiro de Redes de Computadores*, Gramado, RS.
Manjeshwar, A. and Agrawal, D. (2001). Teen: A routing protocol for enhanced effeciency in wireless sensor networks. In *1st International Workshop on Parallel and Distributed Computing Issues in Wireless Networks and Mobile Computing*.
Meguerdichian, S., Koushanfar, F., Potkonjak, M., and Srivastava, M. B. (2001). Coverage problems in wireless ad hoc sensor networks. In *IEEE INFOCOM - Annual Joint Conference of the IEEE Computer and Communications Societies*, pages 1380–1387.
microLinux (2003). Survey about micro linuxes. `http://www.tldp.org/HOWTO/Laptop-HOWTO-18.html`.
Microsoft Windows CE (2003). `http://www.microsoft.com/windowsce/embedded/`.
Millennial Net (2004). Millennial net: Wireless sensor networks. `http://www.millennial.net`.
Motes, M. (2002). The commercialization of microsensor motes. `http://www.sensorsmag.com`.
μAMPS (2002). μAMPS Projet. `http://www-mtl.mit.edu/research/icsystems/`

uamps.
Nakamura, E. F., Figueiredo, C. M. S., and Loureiro, A. A. (2004). Disseminação de dados adaptativa em redes de sensores sem fio auto-organizáveis. In *Simpósio Brasileiro de Redes de Computadores*, Gramado, RS.
National Science Foundation (2004). Report of the National Science Foundation Workshop on Fundamental Research in Networking. http://www.cs.virginia.edu/\~{}jorg/workshop.
Navas, J. C. and Imielinski, T. (1997). Geocastgeographic addressing and routing. In *Proceedings of the 3rd annual ACM/IEEE international conference on Mobile computing and networking*, pages 66–76. ACM Press.
nesC (2003). Nesc v1.1 language reference. http://nescc.sourceforge.net/.
PalmOS (2003). Software 3.5 overview. http://www.palm.com/devzone/docs/palmos35.html.
Pico (2003). Pico Radio. http://bwrc.eecs.berkeley.edu/Research.
Polastre, J. (2003). B-mac protocol. Technical report, Universidade da California, Berkeley.
Pottie, G. J. and Kaiser, W. J. (2000). Wireless integrated network sensors. *Communications of the ACM*, 43(5):51–58.
Rajendran, V., Obraczka, K., and Garcia-Luna-Aceves, J. J. (2003). Energy-efficient collision-free medium access control for wireless sensor networks. In *Proceedings of the first international conference on Embedded networked sensor systems*, pages 181–192. ACM Press.
Rao, A., Ratnasamy, S., Papadimitriou, C., Shenker, S., and Stoica, I. (2003). Geographic routing without location information. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, San Diego, California, USA.
Ruiz, L. B. (2003). *MANNA: Uma Arquitetura para o Gerenciamento de Redes de Sensores Sem Fio*. Tese de doutorado, Universidade Federal de Minas Gerais.
Ruiz, L. B., Nogueira, J. M. S., and Loureiro, A. A. (2004). *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, volume 1, chapter III: Sensor Network Management. CRCPress.
Ruiz, L. B., Nogueira, J. M. S., and Loureiro, A. A. F. (2003). Functional and information models for the manna architecture. *GRES03 - Colloque Francophone sur la Gestion de Reseaux et de Services*, pages 455–470.
Sankarasubramaniam, Y., Akan, O. B., and Akyildiz, I. F. (2003). Esrt: event-to-sink reliable transport in wireless sensor networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 177–188. ACM Press.
SensorNet (2003). Projeto SensorNet DCC/UFMG: Arquitetura, Protocolos, Gerenciamento e Aplicações em RSSFs. http://www.sensornet.dcc.ufmg.br.
Sheth, A. and Han, R. (2003). Adaptive Power Control and Selective Radio Activation For Low-Power Infrastructure-Mode 802.11 LANs. In *IEEE Workshop on Mobile and Wireless Networks (MWN)*, pages 818–818. Held in conjunction with ICDCS 2003.
Sheth, A., Shucker, B., and Han, R. (2003). VLM2: A Very Lightweight Mobile Multicast System for Wireless Sensor Networks. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1936–1941.
Sohrabi, K., Gao, J., Ailawadhi, V., and Pottie, G. (2000). Protocols for self-organization of a wireless sensor network. *IEEE Personal Communications Magazine*, 7(5):16–27.
Srivastava, M. B., Muntz, R. R., and Potkonjak, M. (2001). Smart kindergarten: sensor-based wireless networks for smart developmental problem-solving enviroments. *Mobile Computing and Networking*, pages 132–138.

Standard Commitee of IEEE Computer Society, L. (1999). Wireless LAN medium access control (MAC) and physical layer (PHY) specification. In *Proceedings of the IEEE Communicaton Magazine*, New York, NY, USA. IEEE Std 802.11-1999 edition.

Stann, F. and Heidemann, J. (2003). Rmst: Reliable data transport in sensor networks. In *Proceedings of the First International Workshop on Sensor Net Protoc ols and Applications*, pages 102–112, Anchorage, Alaska, USA. IEEE.

Su, W. and Akyildiz, I. (2002). A Stream Enabled Routing (SER) Protocol for Sensor Networks. In *Med-hoc-Net*.

Su, W. and Akyildiz, I. (2003). QoS Routing (QSR) Protocol for Sensor Networks With Heterogeneous Traffic. In *Revised for Publication*.

Tanenbaum, A. S. (2003). *Computer networks*. Prentice Hall PTR, 4th edition edition.

TinyOS (2004). Tinyos: A component-based os for the networked sensor regime. `http://www.tinyos.net`.

TR 1000 (2004). ASH Tranceiver TR1000 data sheet. `http://www.rfm.com`.

van Dam, T. and Langendoen, K. (2003). An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the first international conferenceon Embedded networked sensor systems*, pages 171–180. ACM Press.

Vieira, L. F. M. (2004a). Middleware para sistemas embutidos e rede de sensores. Master's thesis, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte-MG, Brasil.

Vieira, M. A. M. (2004b). Embedded system for wireless sensor network. Master's thesis, Departamento de Ciência da Computação, Universidade Federal de Minas Gerais, Belo Horizonte-MG, Brasil.

Vieira, M. A. M., da Silva Junior, D. C., Jr., C. N. C., and da Mata, J. M. (2003). Survey on wireless sensor network devices. In *IEEE Conference on Emerging Technologies and Factory Automation*.

Vuran, M. and Akyildiz, I. (2003). Spatial correlation-based collaborative medium access in wireless sensor networks.

VxWorks 5.4 (2003). Datasheet. `http://www.windriver.com/products/html/vxwks54_ds.html`.

Wan, C.-Y., Campbell, A. T., and Krishnamurthy, L. (2002). PSFQ: a reliable transport protocol for wireless sensor networks. In *Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*, pages 1–11. ACM Press.

WINS (2003). Wireless Integrated Network Sensors (WINS). `http://www.janet.ucla.edu/WINS/`.

Woo, A. and Culler, D. E. (2001). A transmission control scheme for media access in sensor networks. In *Mobile Computing and Networking*, pages 221–235.

Ye, W., Heidemann, J., and Estrin, D. (2002). An energy-efficient mac protocol for wireless sensor networks. In *Proceedings of the IEEE Infocom*, pages 1567–1576, New York, NY, USA. USC/Information Sciences Institute, IEEE.

Yu, Y., Govindan, R., and Estrin, D. (2001). Geographical and energy aware routing: A recursive data dissemination protocol for wireless sensor networks. Technical Report UCLA/CSD-TR-01-0023, UCLA Computer Science Department.